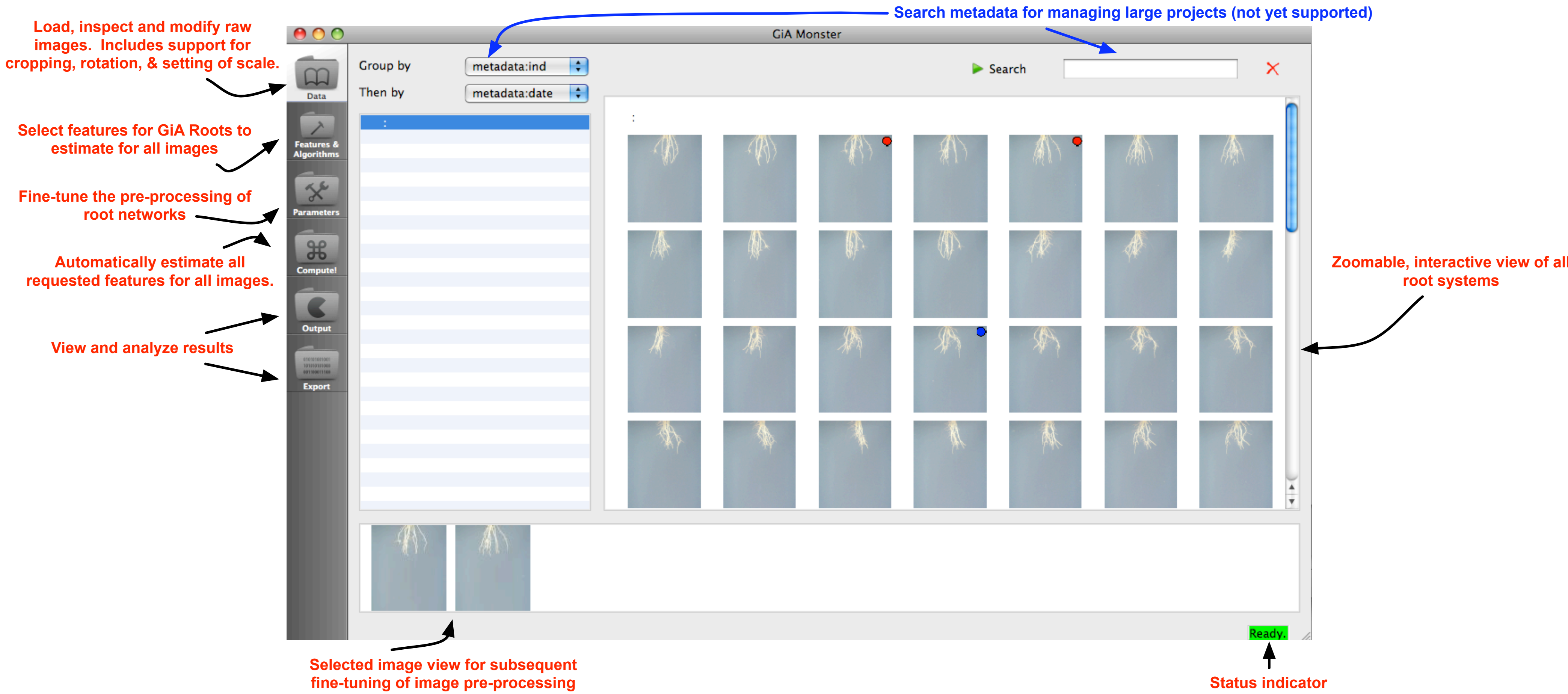
GiA Roots Manual

How to cite:

GiA Roots: Software for the High Throughput Analysis of Plant Root System Architecture (manuscript).
Taras Galkovskyi, Yuriy Mileyko, Alexander Bucksch, Brad Moore, Olga Symonova, Charles A Price, Christopher N Topp, Anjali S Iyer-Pascuzzi, Paul Zurek, Suqin Fang, John Harer, Philip N Benfey and Joshua S Weitz*
*Corresponding author: jsweitz@gatech.edu

Website: http://www.giaroots.org



Search metadata for managing large projects (not yet supported)
GiA Monster
Load, inspect and modify raw images. Includes support for cropping, rotation, & setting of scale.
Data
Group by
metadata:ind
Then by
metadata:date
Search
Select features for GiA Roots to estimate for all images
Features & Algorithms
Fine-tune the pre-processing of root networks
Parameters
Automatically estimate all requested features for all images.
Compute!
Output
View and analyze results
Export
Zoomable, interactive view of all root systems
Ready.
Selected image view for subsequent fine-tuning of image pre-processing
Status indicator

Contents

Chapter 1

Introduction

1.1 What does GiA Roots do?

GiA Roots stands for **G**eneral **I**mage **A**nalysis of Roots. GiA Roots is a software tool to *automatate and facilitate the large-scale analysis of root networks*. GiA Roots has been designed to help scientists and breeders quantify the structure of plant root system architecture, regardless of their prior training in mathematics and computer science.

Like many useful pieces of software (we hope), GiA Roots originated as a question: how can we efficiently analyze many thousands of images of rice grown in artificial substrate? In our case, these thousands of images were from multiple varieties of rice and had been grown in agar gums like that seen in Figure 1.1:

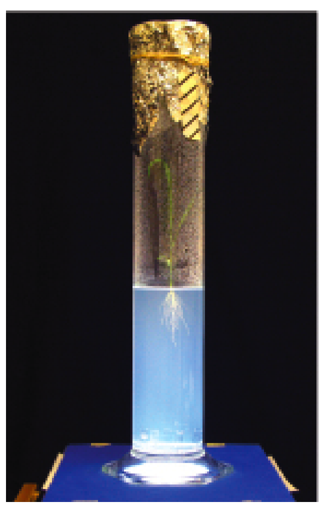


Figure 1.1: Image of a rice plant grown in an artificial gellan gum substrate (Iyer-Pascuzzi et al., 2010).

The rice plant seen in Figure 1.1 was just one of hundreds of plants that our experimental colleagues wanted to analyze. The long-term goal of our collaboration was straightforward: identify genes in the rice genome that contributed to specific modifications of rice root system architecture, e.g., modifying rice roots to improve crop yield in drought or nutrient-stressed conditions. The

rice plants were from a number of different varieties, 12 in our initial pilot study. Comparing genoptyes after 10 days of growth, it was apparent that the genotypes were different from one another (as seen in Figure 1.2). The visual differences between genotypes suggested that aspects of

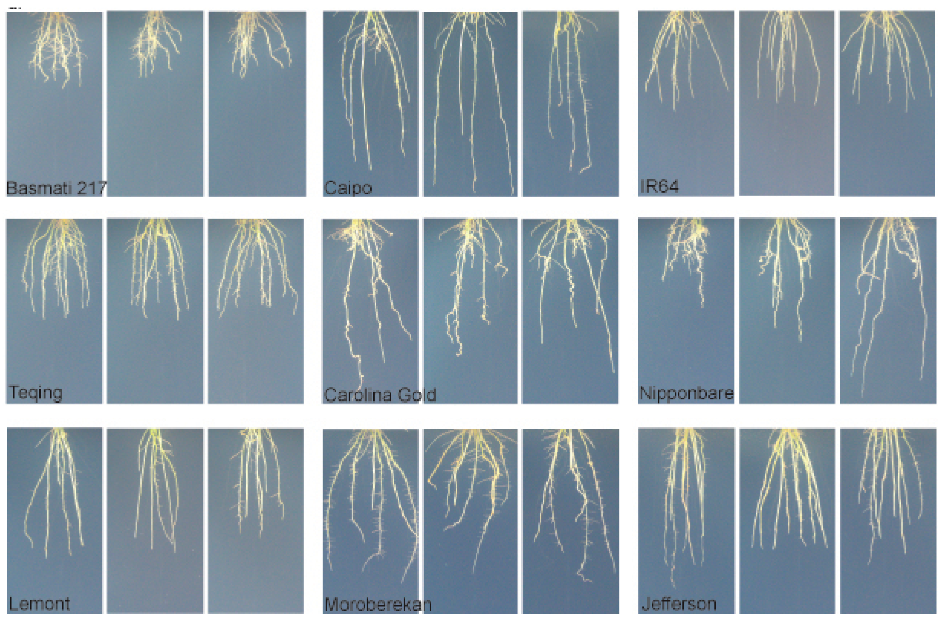


Figure 1.2: Images of 9 rice varietites showing that inter-genotype variation in root system architecture is greater than intra-genotype variation (Iyer-Pascuzzi et al., 2010).

root system architecture could be heritable, and hence, a potential target for breeding. However, visual inspection was insufficient for us to say exactly what was the difference between individuals from two genotypes. For example, it is apparent in Figure 1.2 that Caipo roots are longer than that of Basmati 217. However, are Lemont roots different in any meaningful way when compared to Jefferson?

To make progress toward our goal we had to have some means of *quantifying* rice root system architecture in the laboratory. Hand-scoring of root structure has been used successfully in the field whereas many other groups using ImageJ or other general purpose image analysis software to inspect the properties of individual root networks. We felt there had to be another way. We did not really want to have to go through each individual image, measuring details from root hairs to overall root network properties. Rather, we wanted an automated and efficient software that would:

- Input a large number of images
- Perform the necessary data pre-processing to clean up noise that arose during the imaging step
- Calculate root system architecture features of individual images
- Export all the calculated features for downstream analysis, e.g., statistical analysis of the relationship between genotype, phenotype, and environment.

We found no off-the-shelf solution available. As such, GiA Roots was born, first as a series of Matlab scripts to accompany an imaging platform (Iyer-Pascuzzi et al., Plant Phys 2010) and now, in its current version, as a stand-alone software.

In practice, GiA Roots presumes that you – the user – has access to images of root networks. Perhaps these networks are from your own laboratory or that of your colleague. Perhaps these networks were downloaded from an online database. Whatsoever the case, you would like to know something about these root networks. For example, what is the average width of roots in the network, or what is the volume in the network? At the heart of GiA Roots are algorithms that transform the root systems in each image into a black and white image. Next, GiA Roots calculates features of individual root systems (see Figure 1.3 for an example of what these algorithms actually do). GiA Roots has become an essential tool in our collaborative work. We can now analyze

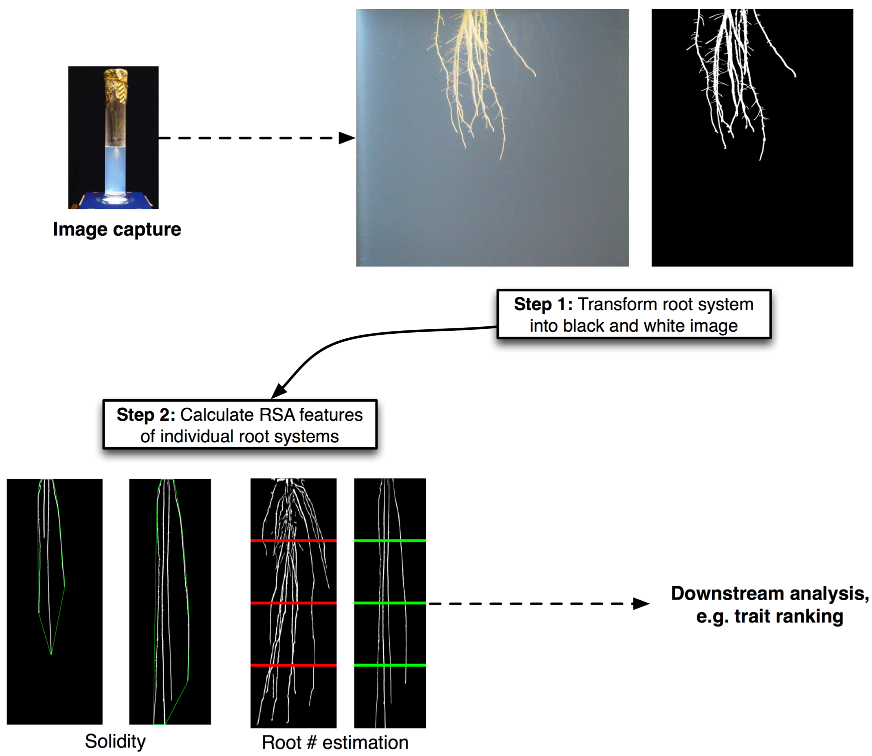


Figure 1.3: What GiA Roots does to each network: (i) transforms images into black and white images; (ii) automatically extracts features of the networks.

thousands of images in a few hours, whereas hand-curated analysis can often require many hours for an individual image.

However, in trying to develop software that would be useful for our colleagues in both academic settings and in industry we were faced with two additional challenges: (i) GiA Roots must be general enough to analyze root images taken in a variety of imaging conditions; (ii) GiA Roots must be extensible so that advanced users could tell GiA Roots to extract features specific to their application. The need for generality means that the user must help GiA Roots pre-process images and decide which features to extract. We have tried to make these steps as intuitive as possible (see Chapter 3). The need for extensibility will be addressed in the advanced sections.

So, in summary, *GiA Roots can*:

- Efficiently estimate quantitative features from images of plant root networks.
- Involve the user in the analysis process without burdening the user with learning technical details of processing.
- Keep track of all intermediate stages in image processing for full reproducibility of feature extraction.
- Permit users to add customized features as part of an integrated analysis pipeline.

GiA Roots accomplishes these goals by combining an intuitive **graphical front-end for all users** with **extensible and robust software architecture for technical users**. Most users will become familiar with the GiA Roots interface, hopefully, in a matter of minutes. Figure 1.4 gives a preview of what the interface looks like.

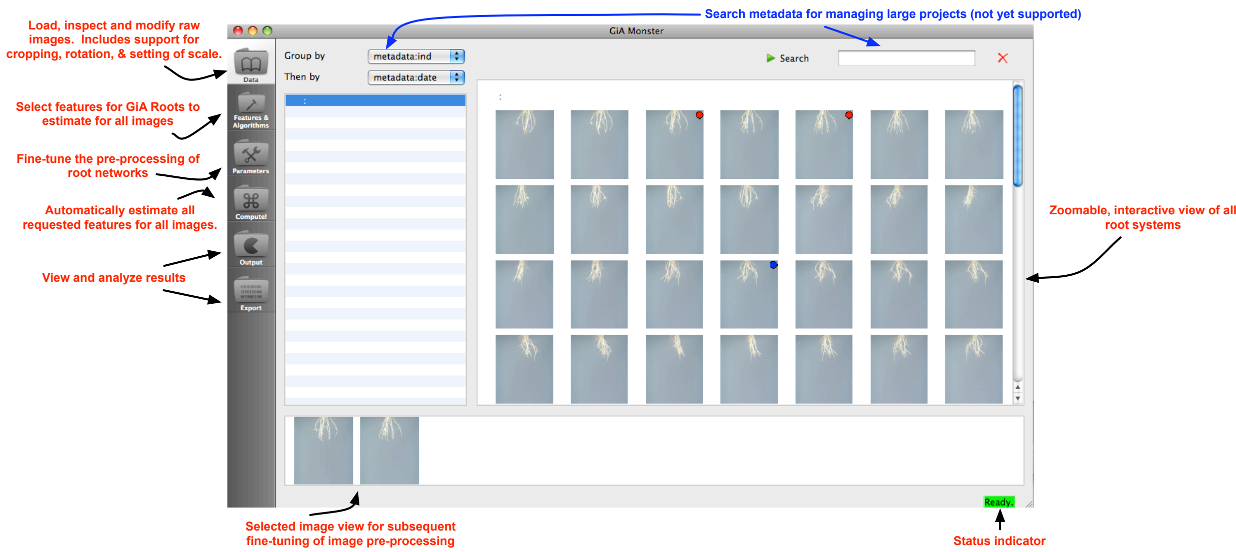


Figure 1.4: Layout of GiA Roots with action buttons on the left, main panel in the center and menus above.

1.2 When should you use GiA Roots?

In deciding when to use GiA Roots, we suggest considering a number of factors. This program is not meant for everyone and for every situation. There are lots of other general and specialized image analysis programs already available, whether freeware, shareware, or commercial. What makes this program different? We developed this software to solve a problem we had in our own research and which, through conversations and discussions with colleages, we realized was a problem shared by many other groups. The problem is: how to take many images of root networks and extract estimates of the shape of root system architecture.

We believe the two most important factors in deciding when to use GiA Roots are the number of images you have and the variability in imaging conditions (see Figure 1.5). For example, GiA Roots was originally designed to analyze the structure of nearly 20,000 images of rice taken from

individuals from over 10 different genotypes of rice grown and imaged in the same conditions. This is an ideal example of a high number of images and low variability condition. It is not important if the networks themselves are highly variable so long as the imaging conditions remain relatively (or ideally exactly) constant.

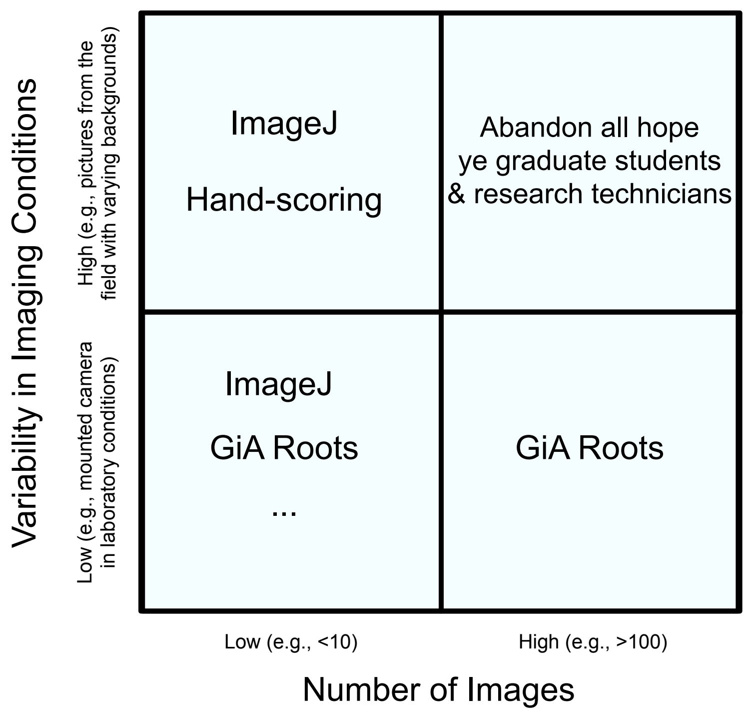


Figure 1.5: Guide to when to use GiA Roots. As a general rule of thumb, GiA Roots excels when faced with analyzing a large number of images of networks taken in very similar imaging conditions.

There are a number of other factors that come into play when deciding to use GiA Roots. We have separated these into advantages and disadvantages.

1.2.1 Advantages of GiA Roots

- Basic and advanced modes
- Reproducibility
- Automatic extraction of features
- Complex feature algorithms built-in
- Extensibility
- Works on image analysis principles

1.2.2 Disadvantages of GiA Roots

- Learning curve for image pre-processing
- Works on image analysis principles

1.3 Organization of this manual

Basic users of the software should read Chapters 1-3 to get an idea of what this software can and cannot, how you can install it, and how you can use it to extract information from images of networks.

Once the software is up and running, Chapters 4-5 will be key to all users to understand how to use the functionality of GiA Roots in pre-processing images and selecting features.

Finally, advanced users of the software who want to use GiA Roots in a command line format should read Chapters 6-8 to understand how the framework works, what the .xml processing commands are, and how to extend GiA Roots with your own proprietary algorithms.

Additional information about licenses of software used in the creation of GiA Roots is contained in Chapter 9.

1.4 Important caveats and disclaimers

GiA Roots is free for academic use. GiA Roots is available for commerical and governmental use for a licensing fee (at the moment it is in beta testing mode). We make no claims that GiA Roots is perfect and you are advised to use it at your own risk! That said, we've put a great deal of time into optimizing the user experience, including verification of the feature estimation algorithms included in the software. GiA Roots is also a *work in progress*, and we expect to continue to improve the software in months and years to come.

As such, please keep in mind the following list of components which we wanted to include in the current release but are **not currently available** because we were sleepy and needed to see our families.

- Undo: in other words, be careful when cropping images, this cannot be undone.
- 3D: we are actively working on incorporating the capability to extract feeatures of 3D reconstructed root networks (wait for it)
- Topological tools: we are actively working on incorporating topological descriptors of root networks.
- Model-based pre-processing instead of thresholding-based pre-processing.
- Other nice GUI gizmos (there are so many more to add).

Finally, GiA Roots has a number of built-in algorithms to automatically extract a root network from an image. This may not always work because (i) the algorithm is imperfect and simply does a bad job given the image quality available; (ii) the user directs the algorithm to use pre-processing settings that lead to poor performance Please keep in mind that all extraction of features are done

on the thresholded image that GiA Roots calculates. Hence, if the thresholded image looks like it is of bad quality (in the Parameters or Output Window), then features should not be considered reliable. Proceed with caution.

1.5 How to cite GiA Roots?

For now, any use of this software should strictly be for beta testing and should not be used in any publication or product development (yet).

In the future, the software and manuscript should be cited as:

GiA Roots: Software for the High Throughput Analysis of Plant Root System Architecture (manuscript).
Taras Galkovskyi, Yuriy Mileyko, Alexander Bucksch, Brad Moore, Olga Symonova, Charles A Price, Christopher N Topp, Anjali S Iyer-Pascuzzi, Paul Zurek, Suqin Fang, John Harer, Philip N Benfey and Joshua S Weitz*
*Corresponding author: jsweitz@gatech.edu

Also: users of the software should also cite: http://www.giaroots.org.

Chapter 2

Quick Start

2.1 More Info

Additional information and the current version of GiA Roots can be obtained from the official web page `http://www.giaroots.org`. Further support is provided by emailing to `gia-roots@biology.gatech.edu`. You are encouraged to post information about encountered bugs while using the software to our bug tracking system Bugzilla at `http://www.rootnet.biology.gatech.edu/bugzilla/`.

2.2 Installation

GiA Roots can be downloaded from the page `http://www.giaroots.org` in the form of the binaries for Windows (XP, Vista, 7) and Mac OS (10.5, 10.6) operating systems. On the page you will be asked to provide your email. GiA Roots is free for all academic use. Academic users are highly encouraged to provide their name, institution and email to help us track the usage of the software. Commerical users must contact `jsweitz@gatech.edu` for information on how to access the software. You will then be redirected to the download page of the GiA Roots binary distribution.

After unpacking the downloaded package GiA Roots is almost ready to use. The only other required download are the QT libraries available for download from `http://qt.nokia.com/downloads`. Users should download the most recent version of either the **QT libraries** or the **Qt SDK**. Please note, that we do not yet offer an automated installation routine, but GiA Roots will run from any location. Simply open the archived file in a directory of your choosing. All temporary files and results of processing will be stored in the project directory that will be specified while using GiA Roots. To start the application, look for an executable file with a GiA Roots “monster” icon, and double-click it. Unless you are an advanced user you should safely ignore all other files contained in the package.

To follow the step-by-step guide of using the GiA Roots, please continue to Chapter 3.

Chapter 3

Starting on your first project

3.1 Creating a project

When you run GiA Roots for the first time, you will see a splash screen (Figure 3.1): Since this is your first time using GiA Roots you'll want to create your first project. To do so, click on "Create New Project" and choose a directory and name of your first project. Note that GiA Roots automatically creates a directory for your project. So, if you call your project "my_first", then a directory called "my_first" will be created where you specify it using GiA Roots interactive window. Once you are more experienced with GiA Roots you can open existing projects, create additional projects, or use the splash screen to browse help topics in this manual or online.

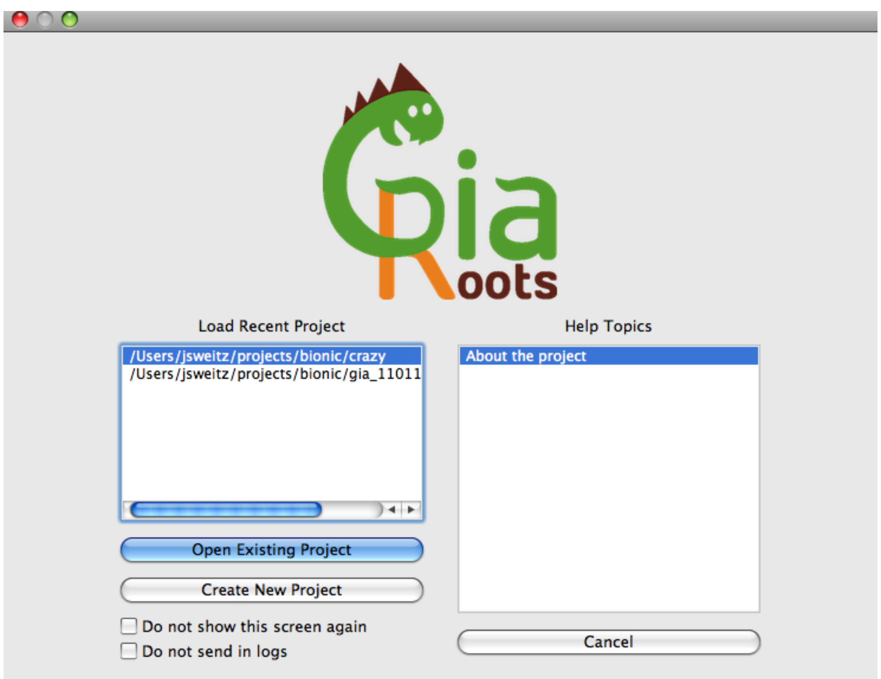


Figure 3.1: GiA Roots splash screen in which you can load an existing project, create a new project, or seek some help.

What is a project exactly? It is:

- Strictly speaking, a directory which contains all the details on the root images you want to analyze, the final results of GiA Roots' analysis, along with all intermediate steps that GiA Roots took in order to estimate root features. Data does not get deleted or replaced from project, so the whole history of computations is being preserved;
- An organized set of images and features accompanied by information that is needed to reproduce them from input data;
- (Potentially) part of a larger analysis pipeline that can be accessed using XML input and output protocols (see Advanced Users section of this manual).

3.2 Loading data

Now that you've named your project, you'll be faced with a (largely) empty workspace (see Figure 3.2). GiA Roots can't help you estimate features of root networks until you specify which images to analyze. In order to import images, click on the Data icon and select "File → Import Images" from the menu at the top. A directory browser will appear and you can select a set of files just as you would in any other Mac, Linux or Windows program.

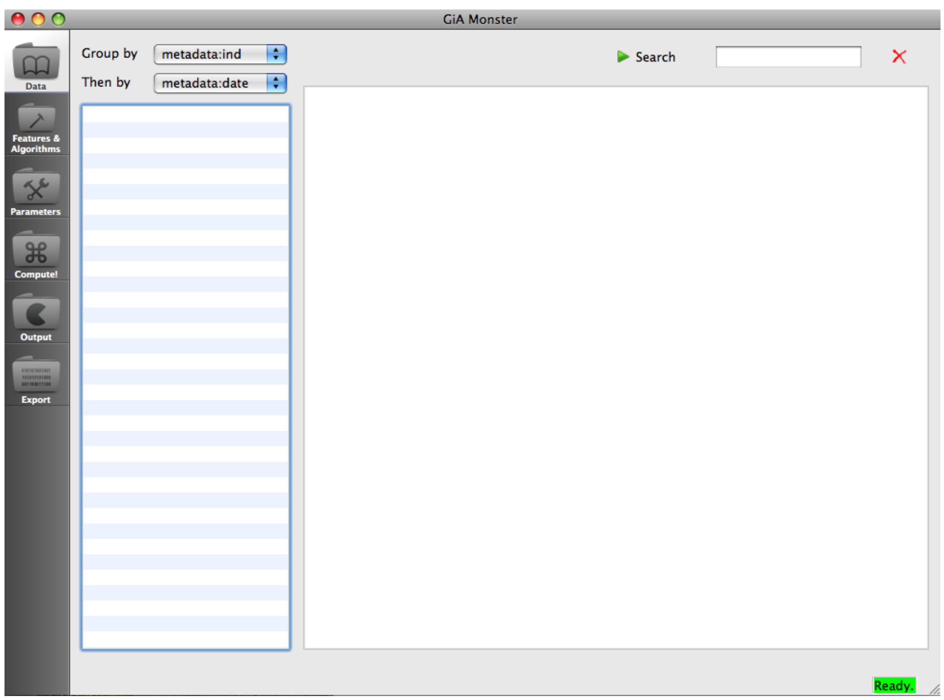


Figure 3.2: GiA Roots is ready to import images - click on the Data icon and then select Import Images from the File menu.

Given that this is your first project, we strongly recommend using a dozen or so images that satisfy the following criterion: (i) they are high-quality with good contrast between foreground root pixels and background non-root pixels; (ii) they are smaller than a few MB in size; (iii) they

are all imaged under (close to) identical conditions. Following this advice will help ensure that you understand the capabilities of GiA Roots even if your actual network analysis needs are more complex. The learning curve for GiA Roots is meant to be fast and the software can handle hundreds, thousands and tens of thousands of images (but you'll need to read more about the software to do such analysis effectively). We have included a set of approximately 200 test images as part of this release.

As images are loaded they will appear in the main window, a red Busy status will appear in the lower right, looking much like that in Figure 3.3. The program is busy creating thumbnails of your images for rapid display, so loading time can vary from a few seconds to minutes depending on the number of images and their file sizes (see Figure 3.4). Once the images are loaded, you may see a

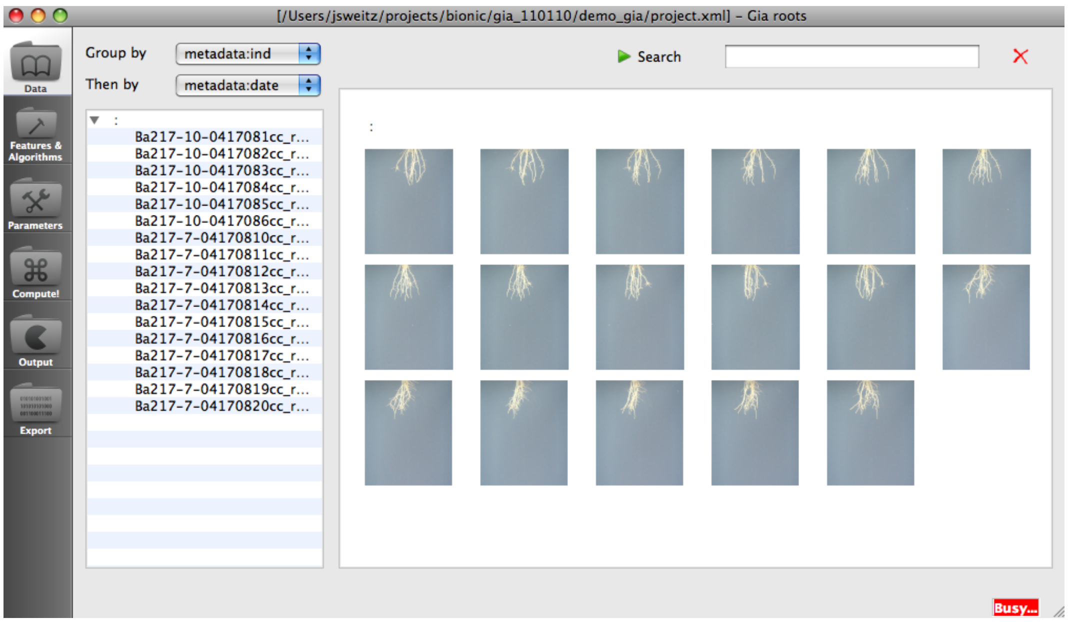


Figure 3.3: GiA Roots is busy loading images.

scroll window on the side, allowing you to scroll through the images you've added to the project. Note that the next time you load your project, the images will appear *much much* faster. Try it!

3.3 Viewing data

Individual images can be selected by double-clicking on them in the main window. Doing so will provide a zoomed-in view of the root network (see Figure 3.5). What are some of the things you can do to an individual images at this point?

- Browse through images one by one
- Zoom in and out of images using your mouse
- Set the scale so that all estimated features can be in units of mm or mm^2 etc., rather than pixels, pixels^2, etc.
- Crop the image(s)

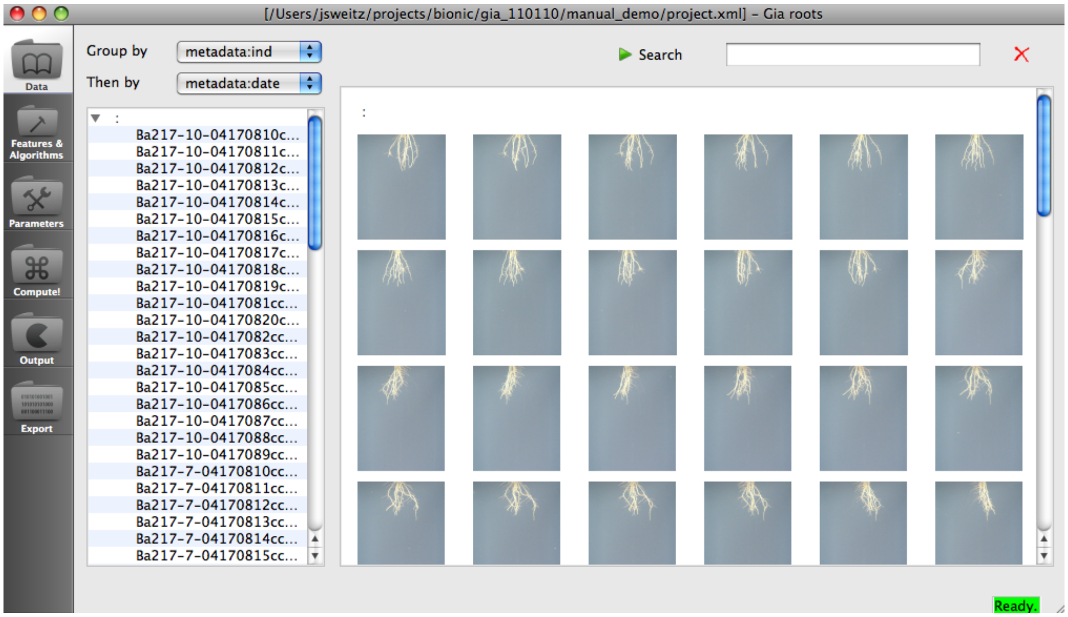


Figure 3.4: GiA Roots is done loading images, and is ready for interactive selection of features and processing options.

- Rotate the image(s)
- Adjust the color of the image(s)
- Add an image to the test list
- Remove an image from the test list

We explain each of these tasks in Chapter 4, but don't let that stop you from exploring......

Done yet? Not quite........

Okay - good. Now that you've had a chance to see that GiA Roots has loaded the images correctly, click on the icon on the left side of the screen labeled "Features & Algorithms".

3.4 Selecting features

By clicking on the icon labeled "Features & Algorithms" you will be taken to a new window in which all of the features that GiA Roots knows how to estimate from a root image are displayed. For example, in Figure 3.6, a zoomed-in view is shown in which one trait is selected (Average root width).

At the moment, the current list of features is (in alphabetical order):

- Average root width
- Ellipse axes ratio
- Major ellipse axis

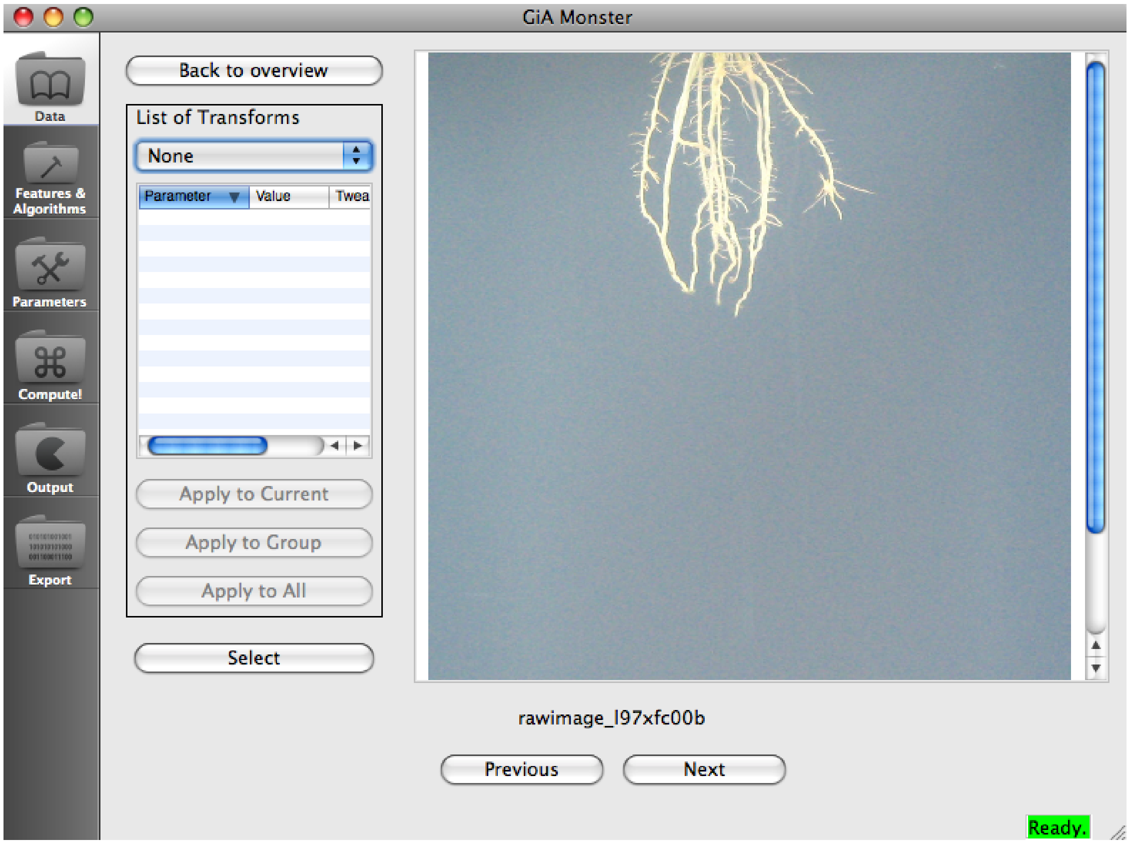


Figure 3.5: Zoomed in view of a root network. Transformations can be applied on the left, zooming and moving is possible with the mouse, and selection of an image to the “marked list” is described in the next chapter.

- Maximum number of roots
- Median number of roots
- Minor ellipse axis
- Network area
- Network bushiness
- Network convex area
- Network depth
- Network length
- Network length distribution
- Network perimeter
- Network solidity

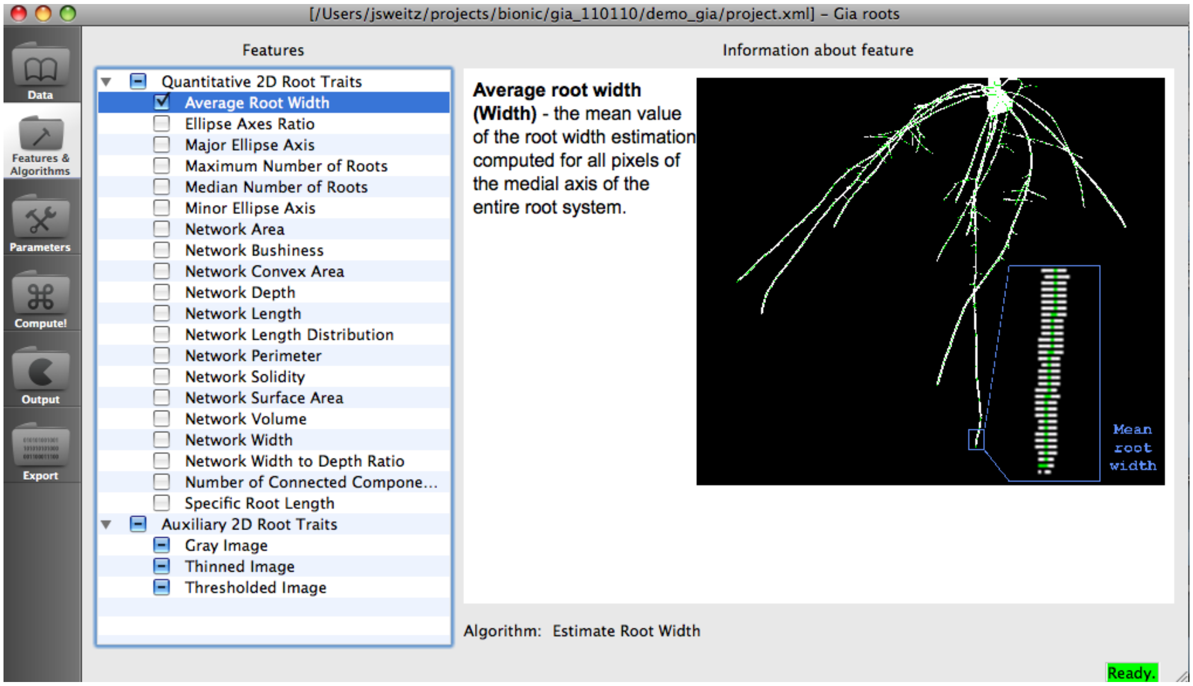


Figure 3.6: The feature and algorithms window allows the user to select features to be estimated for each image. Information about features is displayed on the right.

- Network surface area
- Network volume
- Network width
- Network width to depth ratio
- Number of connected components
- Specific root length
- Gray image
- Thinned image
- Thresholded image

Detailed explanation of these features is available in Chapter 5.

Because this is your first project, select everything. This way you can see all of what GiA Roots can currently estimate for a given image.

3.5 Cleaning images - aka "Parameters"

GiA Roots automatically applies a series of pre-processing steps to images prior to feature extraction. In order to do so, it uses default parameters to distinguish root tissue from background -

however, the default parameters are not meant to suit all needs and applications. By clicking on the icon labeled "Parameters" you will be taken to a new window in which you can assist GiA Roots in the selection of optimal pre-processing parameters.

Once you click on parameters, you will see a dual screen (see Figure 3.7), on the left is the original image, and on the right is the gray-scale image (automatically converted using a RGB to gray step). GiA Roots assumes by default that dark pixels are that of the root and bright pixels are that of the background (i.e., non-root). Click on convert RGB to gray to switch this default.

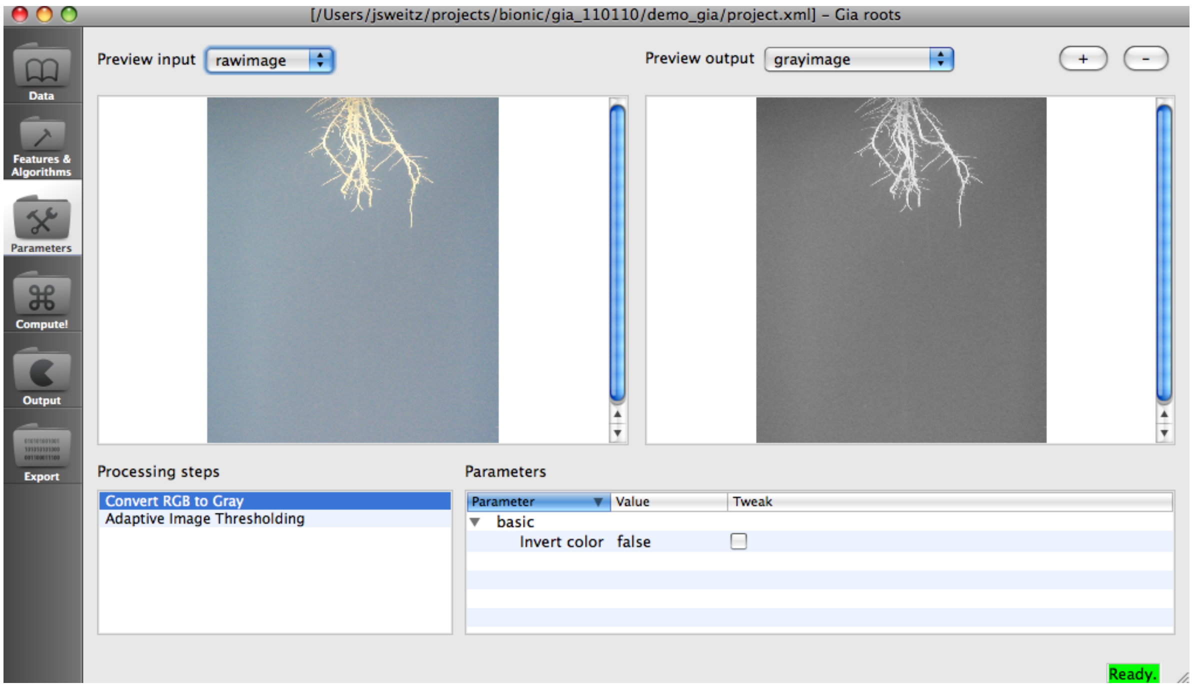


Figure 3.7: The left-right view allows the user to see the pre- and post- of every image processing step, in this case: coverting RGB images to grayscale images.

GiA Roots identifies the root network in the current implementation via the use of a thresholding technique. The default is adaptive thresholding (see next section for more details). Click on the text "Adaptive Image Thresholding" under the list of commands labeled "Processing steps". You should be taken to a new dual screen as seen in Figure 3.8. On the left is the original image, and on the right is the thresholded images. The right image is a single contiguous root network where white pixels are foreground (i.e., network) and dark pixels are background (i.e., not root).

Let's zoom in to find out how well the default parameters did. To zoom in, either click on the + button in the upper right of the window. Or use the middle button of a 3-button scroll mouse (or equivalent) to zoom in and out seamlessly. Here is what we see in the current demo (Figure 3.9):

There are three parameters listed under "basic" for the adaptive image thresholding program. You will eventually get an intuition of how to use these parameters in your favor. In brief,

Block size – denotes the number of pixels in the length of one edge of a sliding window in which adaptive thresholding takes place. Too low a value can lead to holes in your root networks, too high a value and you can pick up spurious root components and/or a halo around your roots.

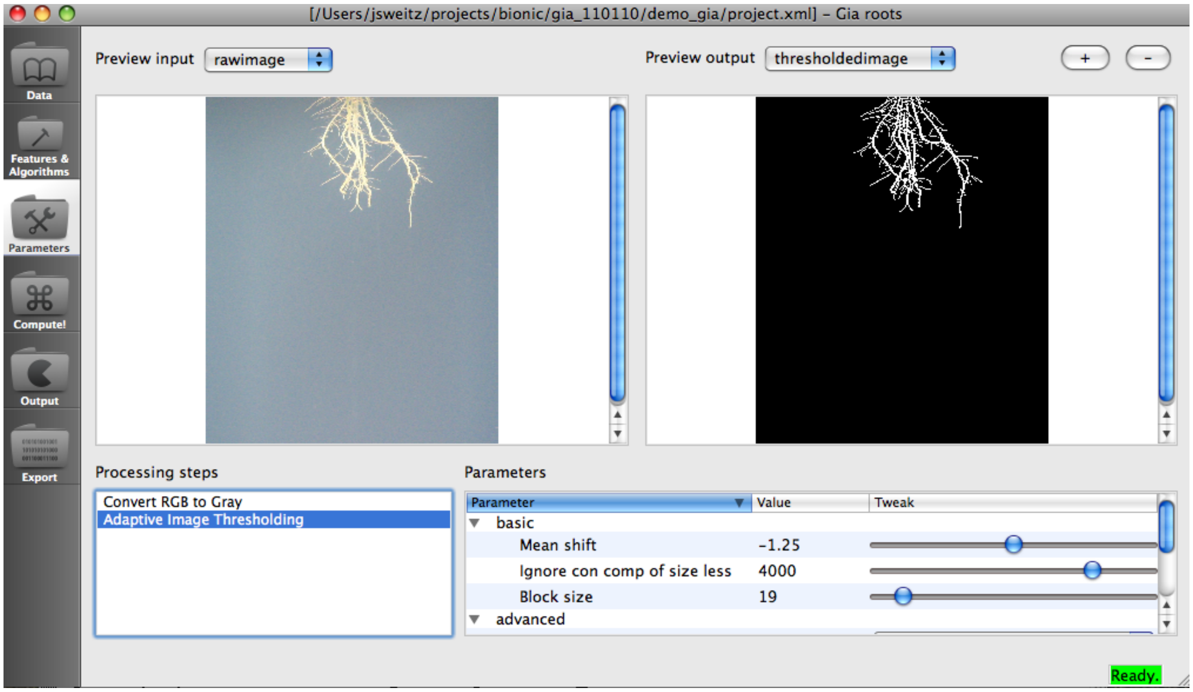


Figure 3.8: In this case, the left-right view allows the user to see the pre- and post- of image thresholding.

Ignore con comp of size less – denotes the minimum number of pixels in a connected component for that block of foreground pixels to be retained. Setting this value too low permits lots of noise to remain in your network (bad for analysis). Setting the value too high may (occasionally) remove a true component of the network which is split from the main network..

Mean shift – denotes the difference between a pixel of interest and the mean pixel intensity of a sliding window before the program considers that pixel to be part of the root. Values less than -1 are generally good, you'll find that higher values lead to noisy networks (which can be compensated by changes in other parameters).

Importantly, you can tweak any of these parameters using the tweak slider, or enter an appropriate number directly into the value field associated with that parameter. Here is what we see in the current demo after further tweaking (Figure 3.10):

Once you are satisfied with your choice, click on the "Compute!" icon to compute your selected features from your images.

3.6 Computing features

The "Compute" page serves as a last step before GiA Roots does the following

- Pre-processes all or some of your images using the pre-processing parameters set in the "Parameters" page.
- Extracts features from all or some of your images.

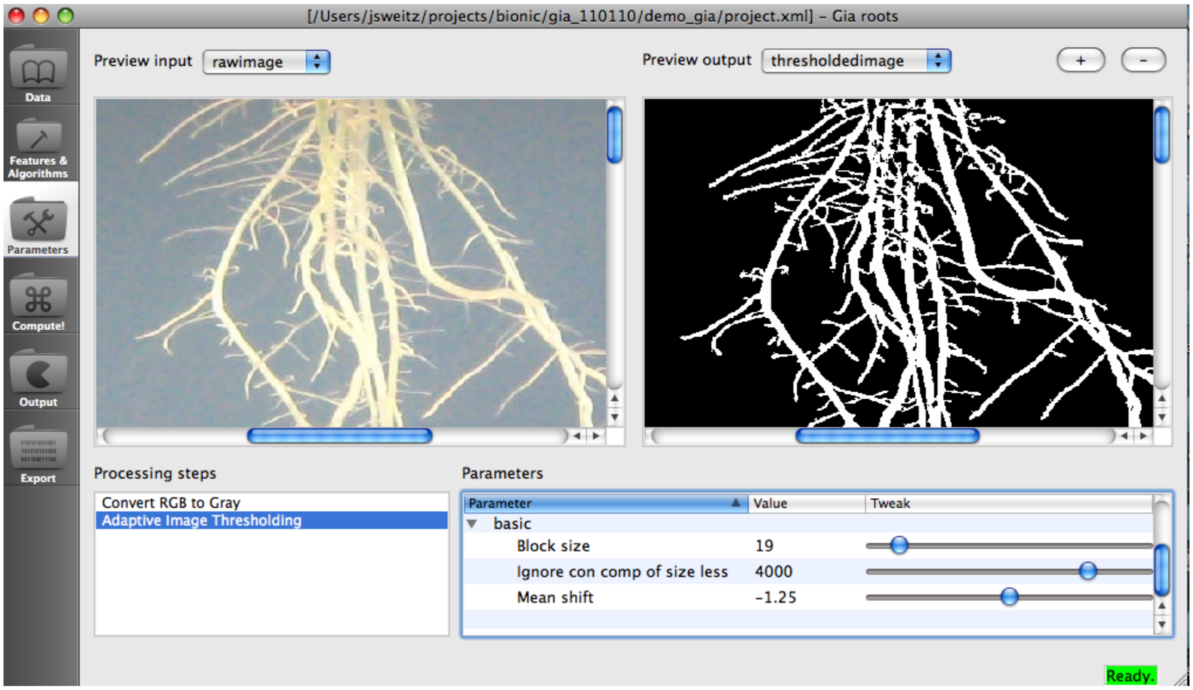


Figure 3.9: The left-right view allows the user to see the pre- and post- of image thresholding. In this case, we have zoomed in significantly and noticed that the overall performance is good, although some small details have been lost in the thresholding process.

- Stores all computed information, which is accessible in the Output and Export pages.

In this demo, 80 images have been selected [1], which will be processed in the pipeline automatically. GiA Roots has also warned us that we have not assigned a scale, but that's okay, we will learn how to do so in a moment.

What next? Click on "Compute on all images" and wait (hopefully not too long) to get your results. It takes GiA Roots approximately 2 minutes on a Mac PowerBook laptop with a 2.33 GHz Intel Core 2 Duo and 4 GB of ram to extract 20 features from this set of 80 images, each of which is approximately 1/4 MB (hence, 20MB in total), (see Figure 3.11).

3.7 Viewing and exporting results

These last two pages ("Output" and "Export") are largely self-explanatory.

Output allows users to view the estimated features in a number of ways. Extracted images can be viewed visually. Extracted quantitative features can be examined on a per image basis, or as distributions (via the "Show distribution" button on the bottom left). Viewing output can be useful to identify outliers in the data (though interactivity will need to be improved in future versions).

Finally, the export option allows the user to export all numerical features and all extracted images to a single file and directory (respectively).

[1] I know, I know, we said don't choose more than 10 for your first time, but trust us, it's not our first time.

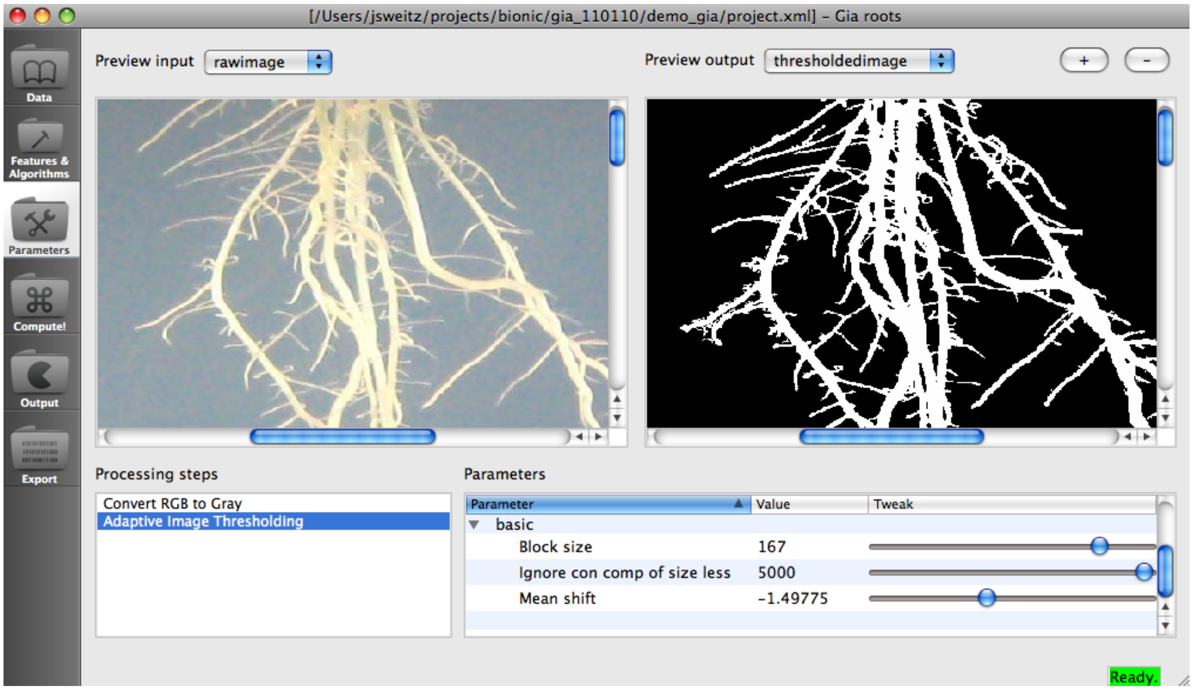


Figure 3.10: The left-right view allows the user to see the pre- and post- of image thresholding. In this case, we have zoomed in significantly and modified the default parameters to increase the solidity of the primary roots.

3.8 What next?

Congratulations for getting this far! Us, developers, should give ourselves a pat on the back since you haven't given up yet.

If you consider yourself a basic user (i.e., you don't want to use command-line tools), then go on to Chapters 4-5 and learn more about ways to get the most out of the software.

If you consider yourself an advanced user (i.e., you do want to use command-line tools), then go to Chapters 6-8 to learn more about the command-line utility. And, if you're really brave, and want to write your own C++ based modules and have them loaded into GiA Roots, then read Chapter 8 a few times.

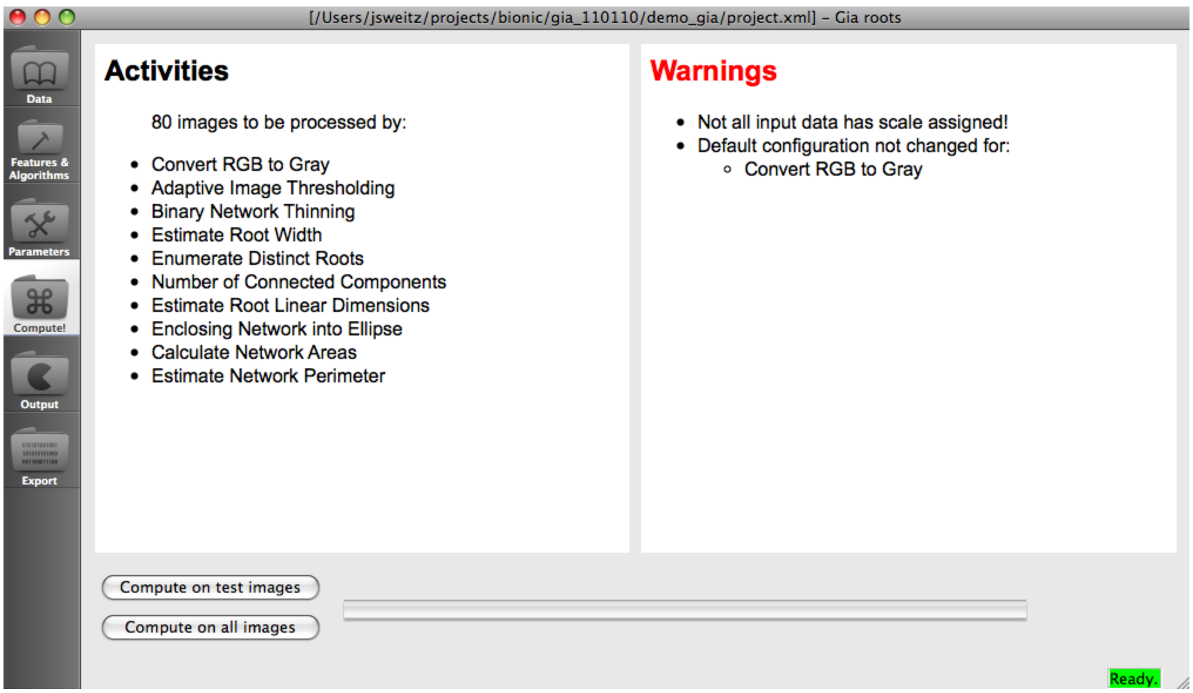


Figure 3.11: The compute page lets you know how many images can be processed, how they will be processed and any warnings that the user should consider addressing.

Chapter 4

GiA Roots Tasks

4.1 Menu

4.1.1 File

The File menu allows users to open projects, create new projects, and import images. These are largely self-explanatory. However, it is important to include a few notes:

Open Project Opening a project will automatically close the current project.

New Project Creating a new project will automatically close the current project.

Import Images This option allows users to add additional images to the project.

Note that there is no menu option that says: "File→ Save Project". The reason is simple: GiA Roots automatically saves all changes dynamically in the course of computations. **If you want to make a backup of your project, archive the entire directory structure of your project.**

4.1.2 Edit

The edit menu provides capabilities related to the testing list. In brief, the testing list is a subset of images selected by the user for evaluation of GiA Roots pre-processing routines. This allows users to check how well GiA Roots extracts networks on small batches of samples prior to running on a full sample. Using the Edit menu, the user can:

- Remove all images in the testing list permanently from the project using the "Edit→Remove test images from project" command. **This cannot be undone - proceed with caution.**
- Deselect all images from the testing list, this does not affect the presence of those images in the project, they remain for continued analysis.

4.1.3 Configuration

The configuration menu allows users to open and save configuration files. A configuration file specifies the current state of the processing commands for the project. By saving a configuration, you can save the set of procedures that GiA Roots should perform, even if you later create a new

project with entirely new images. Careful naming of configuration files can be helpful in speeding workflow. Likewise, when creating a project that has similar image conditions, you can load the configuration.

For example, if you have selected 5 features to estimate, a particular set of thresholding parameters, and a scale, then the configuration file will look something like the following.


```
<?xml version="1.0" ?>
<configuration>
    <algorithm id="%metadata%">
        <property name="%height%" value="1530" />
        <property name="%width%" value="1270" />
    </algorithm>
    <algorithm id="algorithm_manager_defaults">
        <property name="averagerootwidthfeaturelinear" value="feature_root_width_volume" />
        <property name="bushinessfeature" value="feature_roots_number" />
        <property name="ccomponentscountfeature" value="feature_components" />
        <property name="depthfeature" value="features_surface_area_width_depth" />
        <property name="ellipseaxesaspectratiofeature" value="feature_ellipse_axes" />
        <property name="grayimage" value="preparation_color_2_gray" />
        <property name="lengthdistrfeature" value="features_surface_area_width_depth" />
        <property name="majorellipseaxesfeaturelinear" value="feature_ellipse_axes" />
        <property name="maximumnumberofrootsfeature" value="feature_roots_number" />
        <property name="maxwidthfeature" value="features_surface_area_width_depth" />
        <property name="mediannumberofrootsfeature" value="feature_roots_number" />
        <property name="minorellipseaxesfeaturelinear" value="feature_ellipse_axes" />
        <property name="networkareafeatureplanar" value="feature_network_areas" />
        <property name="networkconvexareafeatureplanar" value="feature_network_areas" />
        <property name="perimeterfeaturelinear" value="feature_perimeter" />
        <property name="solidityfeature" value="feature_network_areas" />
        <property name="specificrootlengthfeatureinvplanar" value="feature_root_width_volume
        <property name="surfaceareafeature" value="features_surface_area_width_depth" />
        <property name="thinnedimage" value="thinning_binary" />
        <property name="thresholdedimage" value="thresholding_adaptive" />
        <property name="totallengthfeature" value="features_surface_area_width_depth" />
        <property name="volumefeature" value="feature_root_width_volume" />
        <property name="widthdepthratiofeature" value="features_surface_area_width_depth" />
    </algorithm>
    <core>
        <property name="serialize_image_format" value="jpg" />
        <property name="serialize_to" value="/Users/jsweitz/projects/bionic/gia_110210/new_t
        <property name="thread_pool_size" value="2" />
    </core>
    <algorithm id="gui">
        <property name="targets" value="averagerootwidthfeaturelinear;bushinessfeature;depth
    </algorithm>
    <algorithm id="preparation_color_2_gray">
        <property name="reverse" value="false" />
    </algorithm>
```

```
    <algorithm id="thresholding_adaptive">
        <property name="adaptive_method" value="CV_ADAPTIVE_THRESH_MEAN_C" />
        <property name="block_size" value="50" />
        <property name="max_component_size_to_ignore" value="2500" />
        <property name="subtract_constant" value="-1.85" />
        <property name="threshold_type" value="CV_THRESH_BINARY" />
    </algorithm>
    <algorithm id="thresholding_double_adaptive">
        <property name="block_size" value="15" />
        <property name="drop_value" value="5" />
        <property name="max_component_size_to_ignore" value="4000" />
    </algorithm>
    <algorithm id="thresholding_global">
        <property name="max_component_size_to_ignore" value="4000" />
        <property name="threshold" value="150" />
        <property name="threshold_type" value="CV_THRESH_BINARY" />
    </algorithm>
    <algorithm id="transform_color">
        <property name="invert" value="false" />
    </algorithm>
    <algorithm id="transform_manual_cropping">
        <property name="cropping_height" value="0" />
        <property name="cropping_left" value="0" />
        <property name="cropping_top" value="0" />
        <property name="cropping_width" value="0" />
    </algorithm>
    <algorithm id="transform_rotation">
        <property name="flip" value="none" />
        <property name="rotate" value="0" />
    </algorithm>
    <algorithm id="transform_setscale">
        <property name="metric_scale" value="224" />
        <property name="preview" value="false" />
    </algorithm>
</configuration>
```

Note that there are many defaults here, however, here are the selected parts of this configuration file that highlight the choices made by the user:

```
    <algorithm id="gui">
        <property name="targets" value="averagerootwidthfeaturelinear;bushinessfeature;depthfeatu
    </algorithm>
    <algorithm id="thresholding_adaptive">
        <property name="adaptive_method" value="CV_ADAPTIVE_THRESH_MEAN_C" />
        <property name="block_size" value="50" />
        <property name="max_component_size_to_ignore" value="2500" />
        <property name="subtract_constant" value="-1.85" />
        <property name="threshold_type" value="CV_THRESH_BINARY" />
```

```
</algorithm>
<algorithm id="transform_setscale">
    <property name="metric_scale" value="224" />
    <property name="preview" value="false" />
</algorithm>
```

4.2 Data

4.2.1 Important reminder about transforming images

In order to **commit** any transformation you must either:

- Apply to current (i.e., the transformation will be applied to the current image only)
- Apply to group (i.e., the transformation will be applied to those images currently selected via search or within a given directory of images loaded as part of a bundle).
- Apply to all (i.e., the transformation will be applied to all images).

You must click the apply button to implement a transformation in the GiA Roots pipeline of analysis.

4.2.2 Adding and removing images from a project

Images can be added using File → Import images. To remove images, they must first be selected and added to the test list. Images can be added to the test list by selecting them via a single mouse click on a file name of interest. Removing images from the marked list permanently removes them from the project. **Proceed with caution**.

4.2.3 Setting the scale

GiA Roots can explicitly incorporate a scale for calculation of features. By default, we have set this parameter to be pixels per cm. To access the scale feature, zoom in on an image by clicking on it. Next, select "Set scale" from the List of Transforms. In practice, you can enter this by hand by modifying the Value field. Or, you can click and drag the equivalent of 1cm on the actual image and the program will calculate the scale (Figure 4.1).

4.2.4 Color adjustments

GiA Roots as a default considers root pixels to be bright and backgrounds as black. If your image setup is the reverse, use the Color Adjustments transform and set Invert color to true in the parameters window. For an example of what this does, see Figure 4.2).

4.2.5 Image cropping

Images often have extraneous material in them. To remove borders or other material on the outside of root networks, use the Manual cropping tool from the zoomed mode. Click, hold the mouse down, and then pull out the crop box to the desire size.

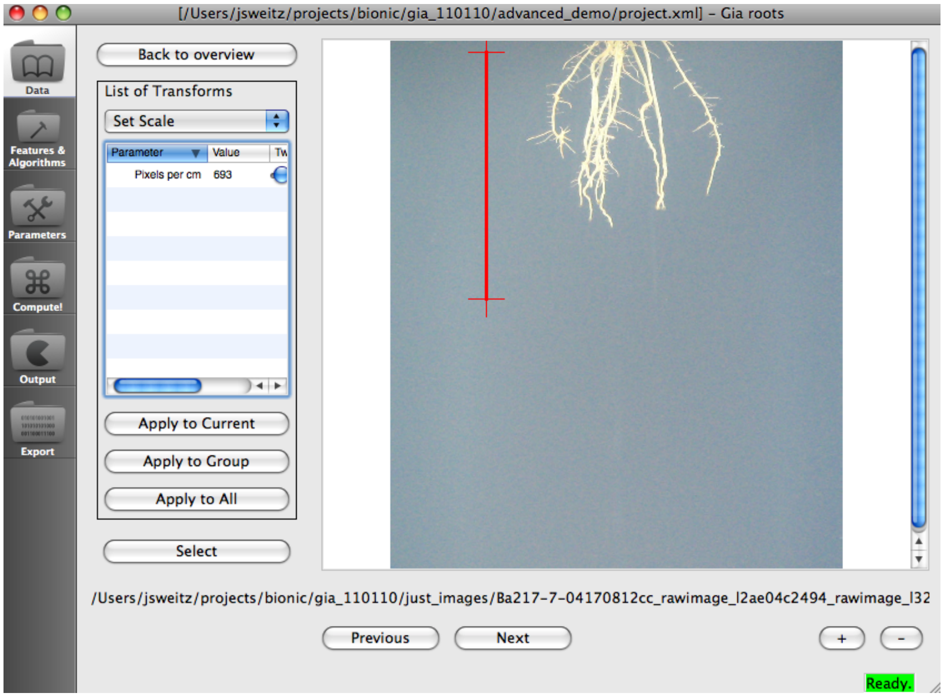


Figure 4.1: Red bar indicates the length of 1cm in a test image. The program will then calculate the scale accordingly (here, 693 pixels per cm), otherwise the user can enter this in directly into the Value field.

4.2.6 Image rotation

Two types of rotations are available: rotations and flips. Rotation is available in 0, 90, 180 and 270. Flips can be horizontal and vertical. These transforms have the standard features of rotation and flip found in many image manipulation packages.

4.2.7 Using testing lists

A testing list is a subset (generally) of the images in a project which the user can then use as tests to see how well the pre-processing and computing pipeline works. Images can be selected for the testing list by clicking on their names from the Data window. In Figure 4.4, you can see an example of what the test image looks like in an example where 2 of 6 images have been selected.

A typical means of using this feature would be to:

- Choose a few images from your image list (perhaps a few from different conditions)
- Select the features of interest
- In the parameters section, try to find a common parameter set for thresholding that will apply equally well to all the test images.
- Compute features on the test images only.
- View the output – if you feel satisfied, setup a larger run. If you are not satisfied, go back to the parameters and continue tweaking until you are satisfied with the output.

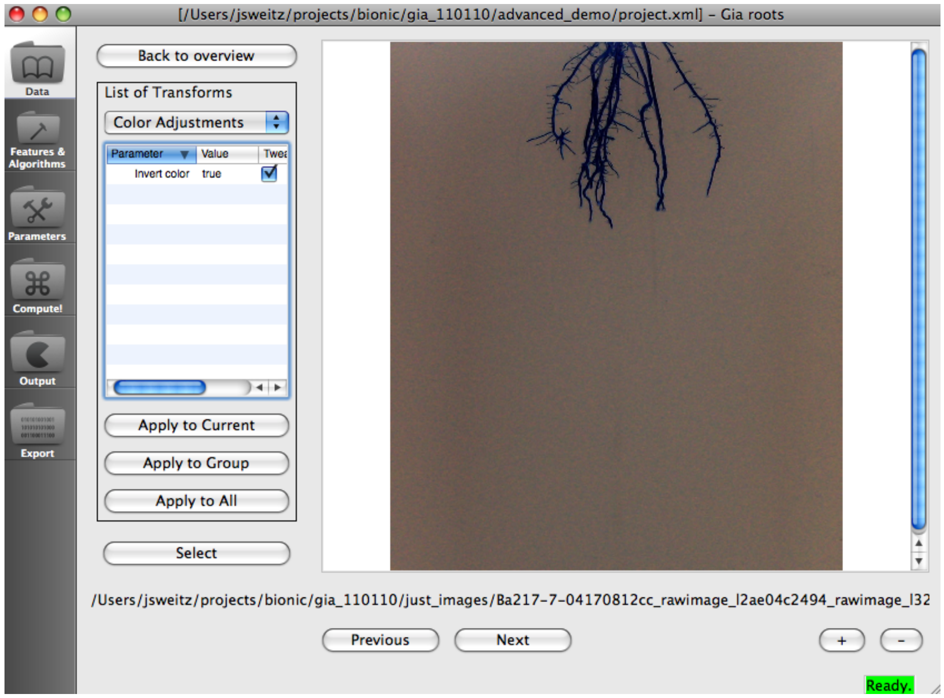


Figure 4.2: Color adjustment flips the colors of the original images.

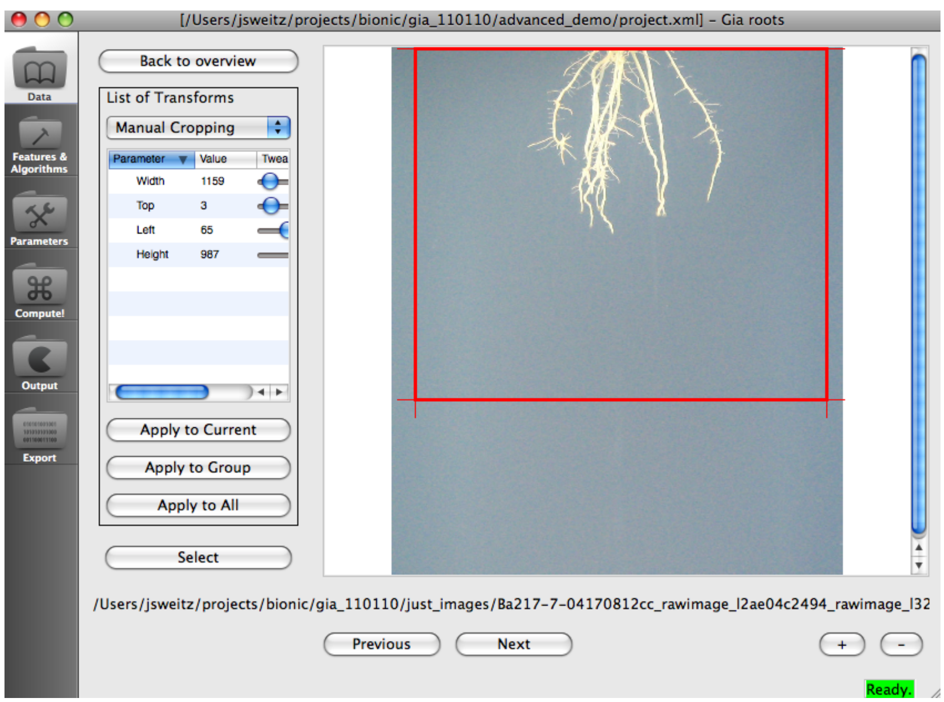


Figure 4.3: A user-selectable rectangle can be used to crop images, the mouse is the best way to do this, just click and drag.

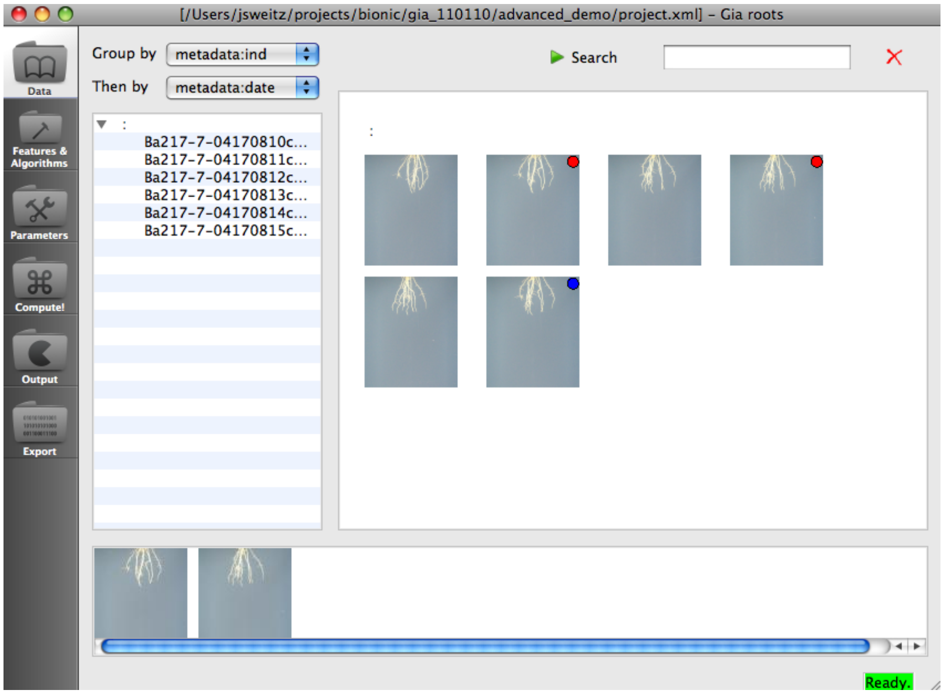


Figure 4.4: Two of six images have been selected for the test list, these same images will be available in the parameters section for interactive pre-processing. The test images are seen in the bottom and marked with red circles.

4.3 Features and algorithms

4.3.1 Selecting features

Although this is largely self-explanatory it is important to point out that GiA Roots optimally computes the path of computation to minimize redundant calculations and also knows when to compute features that you may not deem necessary. As such, even if you only want to know the network volume (for example), you'll notice that GiA Roots automatically creates a gray image, thresholded image and thinned image (all of which it needs to compute volume). Do not be alarmed, this is a good thing.

4.3.2 Selecting a thresholding algorithm

Clicking on thresholded image allows the user to choose among possible methods to accomplish the same goal. The current options are global thresholding, adaptive thresholding and double adaptive thresholding, all of which are explained in the next section.

4.4 Parameters

4.4.1 Global image thresholding

Threshold value – the critical intensity below which pixels are considered to be part of the background, and above which they are considered to be part of the network.

Ignore con comp of size less – denotes the minimum number of pixels in a connected component for that block of foreground pixels to be retained. Setting this value too low permits lots of noise to remain in your network (bad for analysis). Setting the value too high may (occasionally) remove a true component of the network which is split from the main network.

Importantly, you can tweak any of these parameters using the tweak slider, or enter an appropriate number directly into the value field associated with that parameter.

4.4.2 Adaptive image thresholding

There are three parameters listed under "basic" for the adaptive image thresholding program. You will eventually get an intuition of how to use these parameters in your favor. In brief,

Block size – denotes the number of pixels in the length of one edge of a sliding window in which adaptive thresholding takes place. Too low a value can lead to holes in your root networks, too high a value and you can pick up spurious root components and/or a halo around your roots.

Ignore con comp of size less – denotes the minimum number of pixels in a connected component for that block of foreground pixels to be retained. Setting this value too low permits lots of noise to remain in your network (bad for analysis). Setting the value too high may (occasionally) remove a true component of the network which is split from the main network.

Mean shift – denotes the difference between a pixel of interest and the mean pixel intensity of a sliding window before the program considers that pixel to be part of the root. Values less than -1 are generally good, you'll find that higher values lead to noisy networks (which can be compensated by changes in other parameters).

Importantly, you can tweak any of these parameters using the tweak slider, or enter an appropriate number directly into the value field associated with that parameter.

4.4.3 Double adaptive image thresholding

Neighborhood size – denotes the number of pixels in the length of one edge of an expanding radius in which thresholding takes place. Too low a value can lead to holes in your root networks, too high a value and you can pick up spurious root components and/or a halo around your roots. Also note that too high a value can be very slow to compute.

Ignore con comp of size less – denotes the minimum number of pixels in a connected component for that block of foreground pixels to be retained. Setting this value too low permits lots of noise to remain in your network (bad for analysis). Setting the value too high may (occasionally) remove a true component of the network which is split from the main network.

Bound drop value – denotes the strength of a gradient in image intensity for a pixel to be considered part or not part of the root network. Importantly, you can tweak any of these parameters using the tweak slider, or enter an appropriate number directly into the value field associated with that parameter.

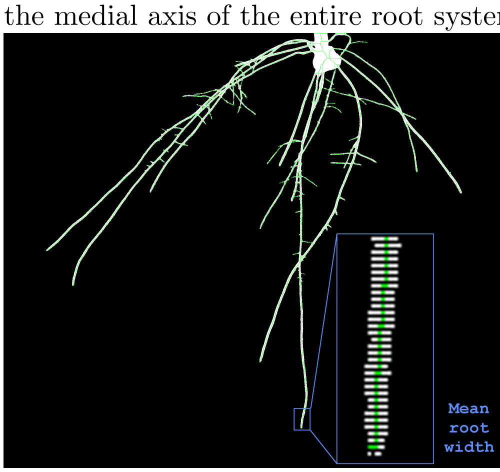
IMPORTANT NOTE: parameters page is designed to find suitable parameters quickly and interactively. The images displayed in the parameter page are thus reduced in size compared to originals. Technically, actual results of thresholding can differ from what you will see while tweaking parameters. Usually the difference can hardly be seen by eye. In order to properly test parameters one more time before executing the whole batch you might want to perform a short run on a testing list of images, as explained in section 4.2.7, in this case results will be absolutely the same as during whole batch processing.

Chapter 5

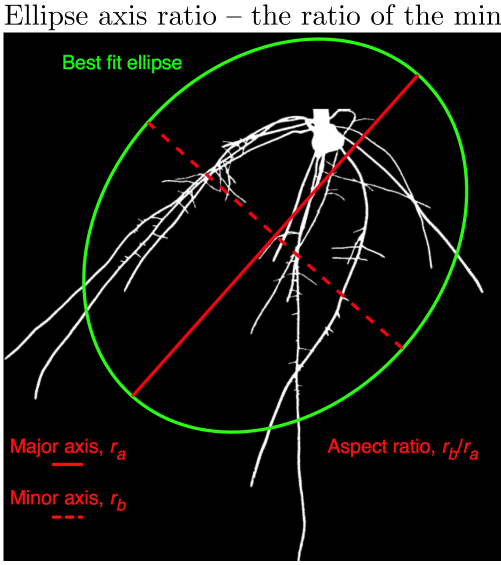
Feature Details

5.1 List of currently supported features

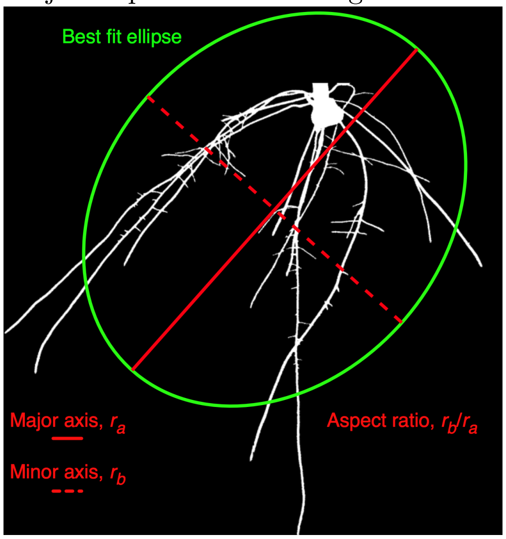
- Average root width – the mean value of the root width estimaton computed for all pixels of the medial axis of the entire root system.



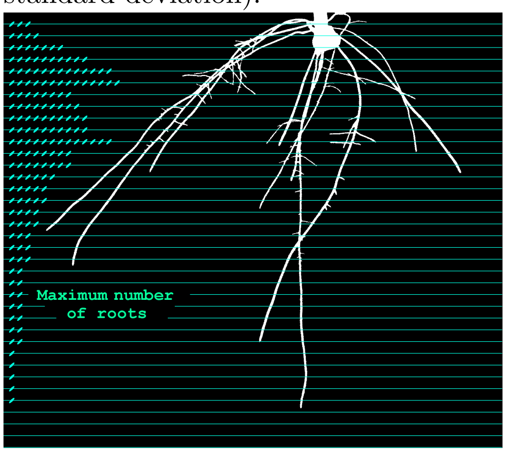
- Ellipse axis ratio – the ratio of the minor to the major axis of best fitting ellipse.



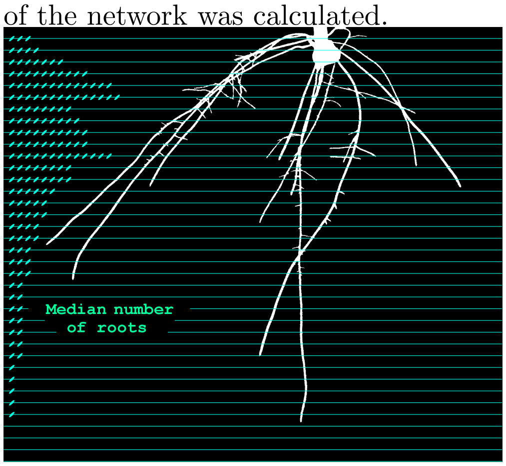
- Major ellipse axis – the length of the major axis of the best fitting ellipse to the network.



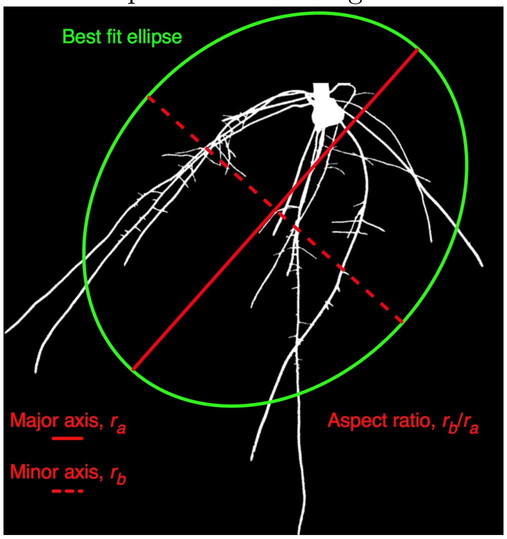
- Maximum number of roots – after sorting the number of roots crossing a horizontal line from smallest to largest, the maximum number is considered to be the 84th-percentile value (one standard deviation).



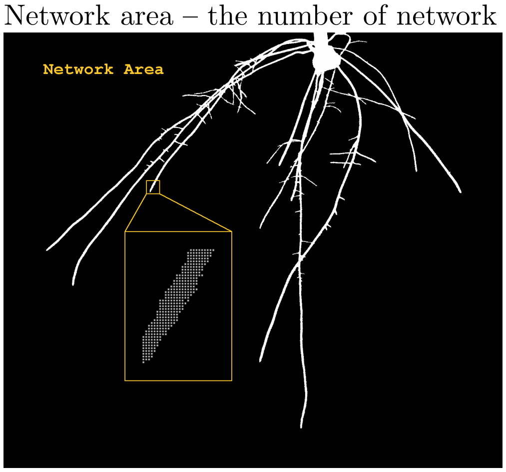
- Median number of roots – the result of a vertical line sweep in which the number of roots that crossed a horizontal line was estimated, and then the median of all values for the extent of the network was calculated.



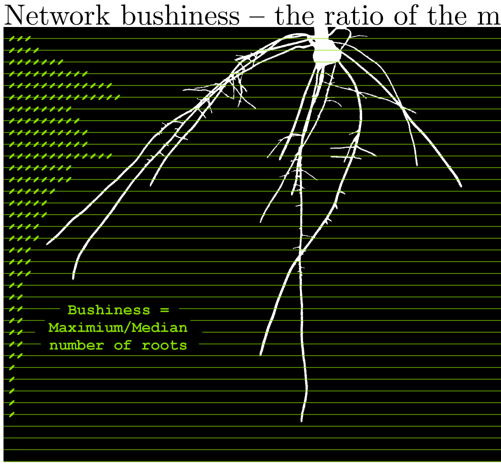
- Minor ellipse axis – the length of the minor axis of the best fitting ellipse to the network.



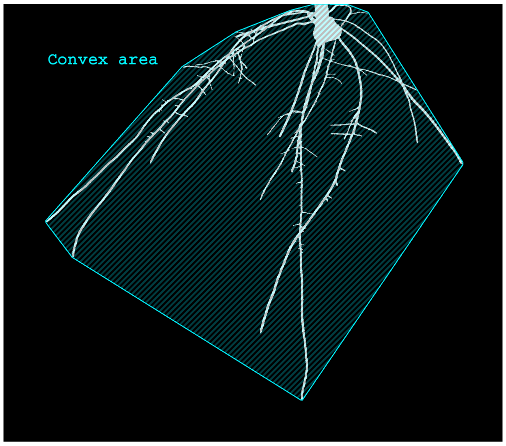
- Network area – the number of network pixels in the image.



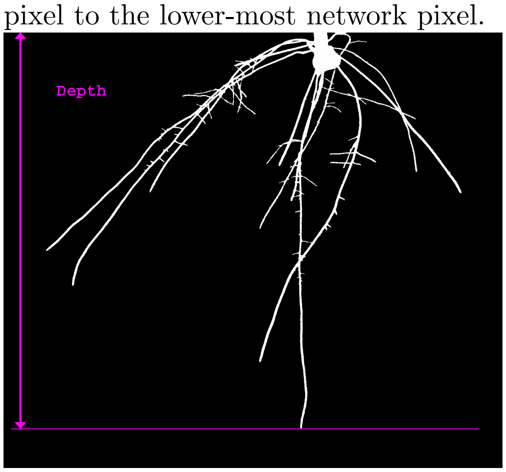
- Network bushiness – the ratio of the maximum to the median number of roots.



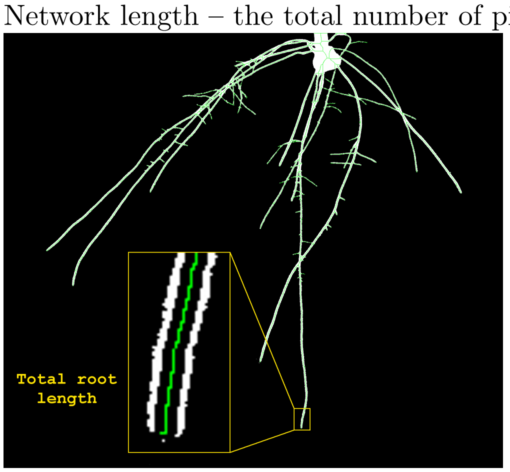
- Network convex area – the area of the convex hull that encompasses the image.



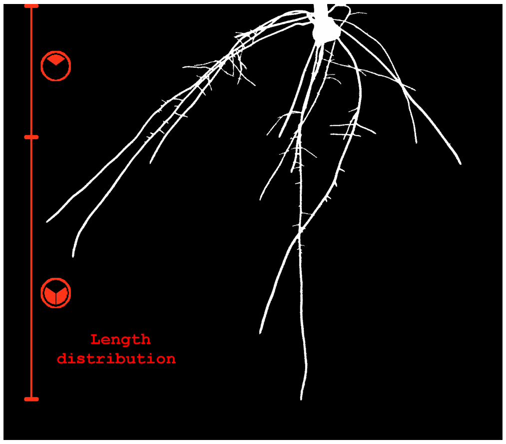
- Network depth – the number of pixels in the vertical direction from the upper-most network pixel to the lower-most network pixel.



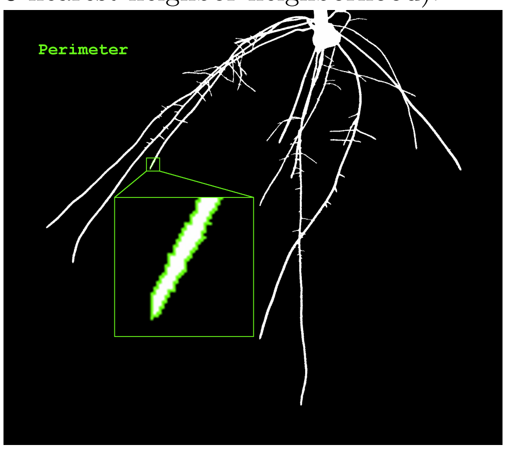
- Network length – the total number of pixels in the network skeleton.



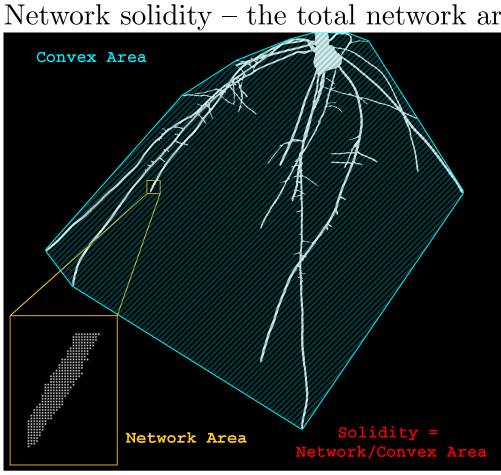
- Network length distribution – the fraction of network pixels found in the lower 2/3 of the network. The lower 2/3 of the network is defined based on the network depth.



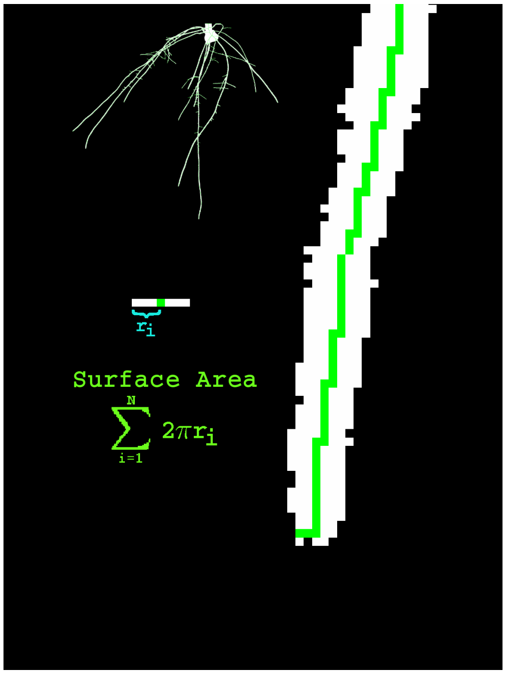
- Network perimeter – the total number of pixels connected to a background pixel (using a 8-nearest neighbor neighborhood).



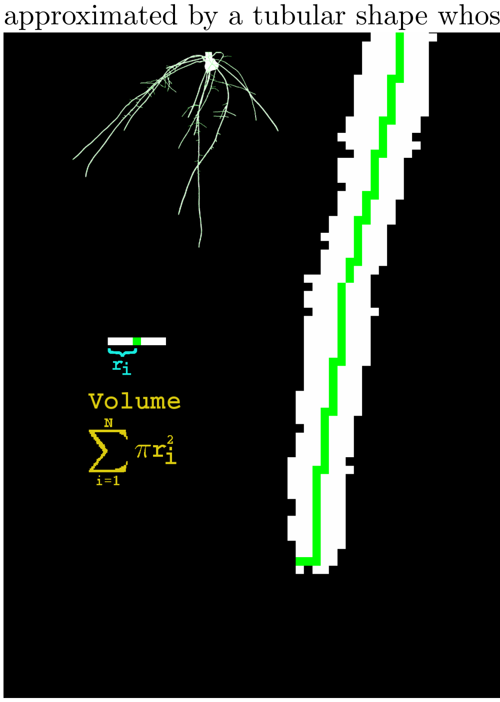
- Network solidity – the total network area divided by the network convex area.



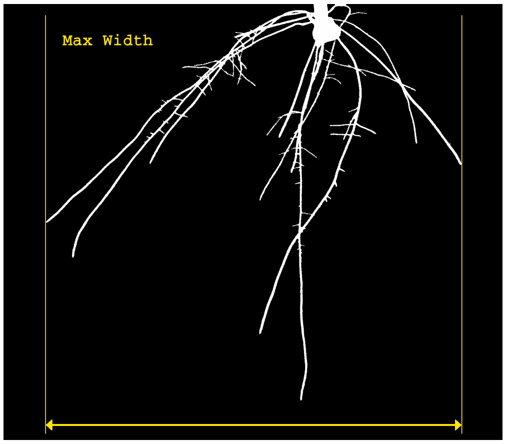
- Network surface area – the sum of the local surface area at each pixel of the network skeleton, as approximated by a tubular shape whose radius is estimated from the image.

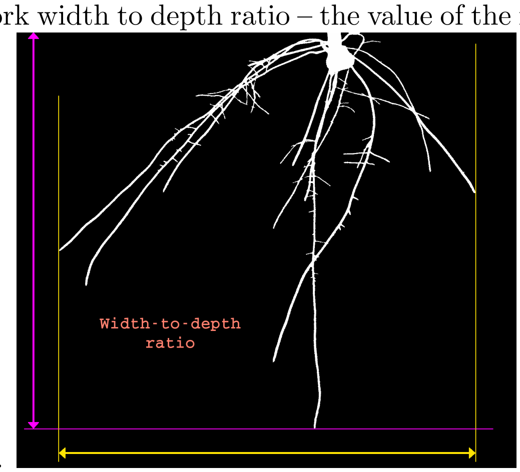


- Network volume – the sum of the local volume at each pixel of the network skeleton, as approximated by a tubular shape whose radius is estimated from the image.



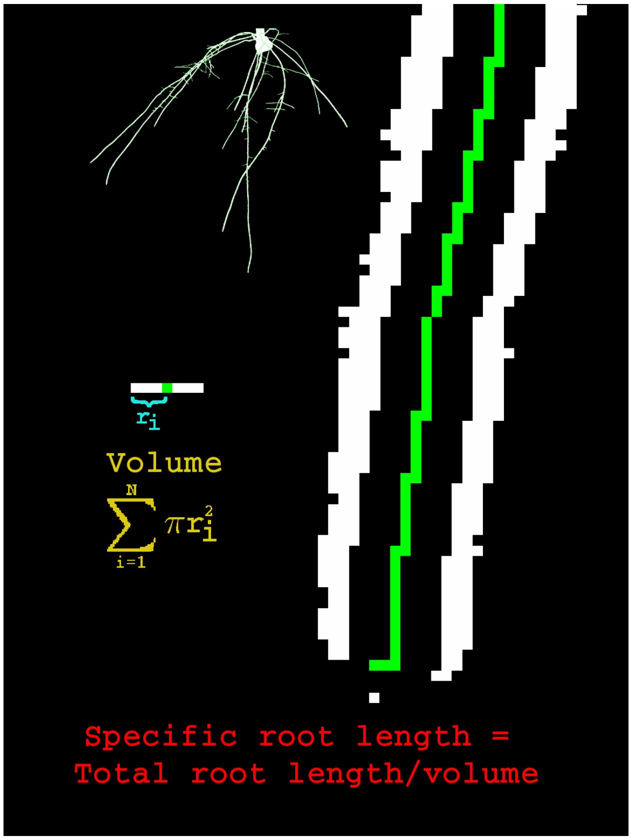
- Network width – the number of pixels in the horizontal direction from the left-most network pixel to the right-most network pixel.



- Network width to depth ratio – the value of the network width divided by the value of network depth. 

- Number of connected components – an integer denoting the number of connected gruops of network pixels in the image after image pre-processing. For example, if all network pixels in the thresholded image are connected to all otheres via a contiguous path of nearest neighbor pixels then the value is 1. If the root network has a break in it somewhere that separates the network into two subnetworks then the value is 2. Note that a real root should only have 1 connected component, but due to errors in image acquisition and pre-processing, the value may be greater than 1 and can be used as a quality control value.

- Specific root length – total root length divided by root system volume. Volume is estimated as the sum of cross sectional areas for all pixels of the medial axis of the root system. The total root length is the number of pixels in the medial axis of the root system.



- Gray image – an image in which the color value of each pixel is a number between 0-255 denoting the intensity (0 is black, 255 is white).
- Thinned image a skeletonized version of the original image which preserves the topology of the original image. For example, one way to determine a thinned imageis to gradually erode an image from its boundaries until all that is left is a single-pixel wide skeleton.
- Thresholded image – is a result of a complex procedure in which "object" pixels (in this case, pixels that are part of the root system) are separated from "background" pixels. The result of which is an image in which the foreground (usually white) is separated from the background (usually vlack). This software includes several algorithms for thresholding.

5.2 Features we anticipate to be included in the future

- Explicit network features
- Distribution features
- 3D features
- Topological features

Chapter 6

Advanced

6.1 Package contents

The folder structure of the developer package is the following:

- The *giaroots-cmd* executable file in the root of the package represents the tool that can execute batch processing jobs (Windows: interpreter.exe ,MacOSX: interpreter.bin). We will refer to it as the *interpreter* in this manual. See Section 6.4 for instructions on how to use it.
- Root folder of the package also contains several examples of jobs for GiA Roots (they have a "job" prefix). To provide parameters for various algorithms to be used for execution, GiA Roots uses configuration files, (in examples they have "config" prefix). We separate configuration from jobs because jobs adress a particular image set, while the same configuration of processing steps can be succesfully applied to several image sets.
- Pluggable modules are situated in the *algorithms* subfolder. This subfolder can have several levels of hierarchy. The interpreter will look automatically for every module inside the algorithms subfolder. A number of precompiled algorithms are included to demonstrate functionality of the framework. (see paper)
- Subfolders that will be used exclusively for developing custom algorithms are *include* and *lib*. In *example-histogram* subfolder a ready-to-go example of a custom algorithm is provided. To get information needed to compile and use this algorithm proceed to the section 8.
- Job descriptions are provided in files with extension *job*. To execute a specific job, the user needs to supply its filename as a startup parameter of the *interpreter*.

6.2 Troubleshooting

One of the goals of this software is to hide as many technical details from the user, such that users can focus on the science of analyzing RSA rather than the mechanistic details of extracting root features. The user finds assistance for solving technical problems in the log-file (gia.log), which contains the validity checks for every execution step in GIA Roots. In the log-file errors are marked as `ERR` and appended by a short error description. Logs can also be used to check the performance of the processing pipeline. Timestamps of for every processing step are provided here.

6.3 Common mistakes

Issue	Reason + Fix

giaroots would not start. `"got response (0) from server;`
`waiting for (90565419)`
`You need to be online to be able to run`
`interpreter, sorry for inconvenience."`
	giaroots can't connect to the license server. Check the internet connection and repeat the procedure. If internet connection is ok, you can manually delete `gia.log` file in the folder where gia executable is situated.

giaroots does nothing. `"Specify job file as a first parameter for` `the daemon"`	Interpreter requires a command line parameter – path to job file to execute
giaroots does nothing. `"Can not read (test-job.xml) file (not valid` `XML)"`	Job file specified as a parameter does not exist or it is not a well-formed XML. Try to open the file with external XML editor to check for errors.

giaroots does nothing. `"Library (./somealgo.so) doesn't have`
`a factory function.  See documentation.`
`(dlopen(./somealgo.so, 2):`
`Library not loaded:  libcv.dylib`
`Referenced from:  ./somealgo.so`
`Reason:  image not found)"`
	Dynamic libraries referenced by algorithms were not found. Please consult section 6.4 on how to correct this.

giaroots does nothing. `"0 libraries to check` `(0)total; known types:"`	Folder with algorithms was not found. Verify that you are running interpreter from framework root directory, check existence of `algorithms` folder.
No images uploaded. `"No images found in the` `[./puttestimageshere/] folder"`	Check the path of the folder you are uploading images from. Sometimes problem might get fixed by using global path instead of relative.
giaroots does not execute directives. `"Unknown type rawimage.  Check if all needed` `plugins are loaded."`	All or some of the algorithms were not loaded, so that some of the data stored in the project can not be properly read. Please consult section 6.4 on how to correct this.

6.4 Interpreter usage

If you are using windows, then you should be able to execute interpreter without any further configuration or changes. Be sure that root folder of the developer package is the working directory for the interpreter (do not run it like this *c:/myfiles/gia-test/giaroots-cmd.exe*).

If you are using MacOS or Linux, you must go to the root folder of the developer package and execute the following command in the shell:

- *export LD_LIBRARY_PATH=./lib/* in case of Linux;
- *export DYLD_LIBRARY_PATH=./lib/* in case of MacOS.

After such operations you should be able to execute the *interpreter*.

Developer package ships with several demo jobs (together with pictures) that should produce meaningful results. You might want to play with parameters used to process sample images by changing values in the configuration files as described in section 7.4.

Chapter 7

Advanced: Technicalities

Formats of files used in the framework are detailed below.

7.1 Job file

The interpreter accepts job files in xml format that contain directives guiding the behavior of the framework in a sequential manner. The storage driver is specified separately to indicate the type of storage used to hold the project files. Currently only XML format of projects is supported, with a Database implementation under consideration for future releases.

Again, all directives found in job file are applied to each of the datasets in the project independently and possibly simultaneously, so we can assume that the state of all datasets is the same after completion of planned jobs.

7.2 Directives

7.2.1 job

Attributes:

- `config` Default configuration to use.
- `driver` Storage driver to load/save project
- `connection_string` Connection string to pass to the storage driver to establish connection to the project

This is the root directive of the job file. It defines the project we are working on and the default configuration to use during processing.

```
<job config="extras/config-test.xml" driver="xml" connection_string="./proj-test">
...
</job>
```

7.2.2 upload

Attributes:

- `connection_string` Connection string to load images from.

adds images found at path specified by `connection_string` attribute to the project. Copies all images and creates framework project if not existed before. It will recursively traverse all of the subfolders of the specified path. Framework will distinguish images with equal names from different subfolders by their relative path, and in the same way it will be possible to distinguish between them in the final report.

```
<upload connection_string="~/dev/giaroots/data/raw_images_paul_test01/" />
```

7.2.3 export

Attributes:

- `driver` Export driver
- `connection_string` Connection string for outputting the results

establishes an output stream, where results of computations got translated even during the process of computation. Currently framework supports streaming of results in csv format, readable by most statistical packages.

```
<export driver="csv" connection_string="./test-paul.csv"/>
```

7.2.4 compute

Attributes:

- `types` Data types to compute, semi-colon separated
- `config` Optional. Configuration to use while computing these types, has priority above any default configurations loaded

executes algorithms to create the results of required types for each dataset. Framework automatically loads all available algorithms that can be found and establishes an execution workflow of algorithms based on the types of data available in each dataset. This is made possible because each algorithm provides metadata on which types it consumes and produces. However, in case of ambiguity, when several distinct flows are possible (for example due to several variants of algorithms performing the same transition) algorithm manager can pick any of the available paths without guarantees. To guide this process it is possible to explicitly specify for each of the output types, which of the algorithms to use to produce it. See section 7.6 for more information on how to specify default algorithms.

```
<compute types="thresholdedimage;thinnedimage" config="extras/config-test1.xml" />
```

7.2.5 describe

Attributes:

- `data_id` Id of object to infer configuration for
- `filename` Filename to store configuration to

as all of the data objects ever computed inside one project have their metadata stored in the project, there is a possibility to check what configuration values where used to compute existing or older data objects. Based on the provided data object id framework will extract all related configuration values from the project metadata in the same format that can be used as input for the framework itself. This process is recursive, so it will also output configuration values for those data objects that were used as an input to the last step of computation, and so on.

```
<describe object_id="croppedimage_234928dsfsdfok" filename="extras/older-
    configuration.xml" />
```

7.3 Project format

Computation project of the framework is fully contained in one directory. It contains set of images or other binary files representing results of computation and the xml file storing all the metadata that describes the project, all data that is stored inside and information about how this data was computed using framework.

Configuration that was used to compute every data object can be inferred from the information contained in the project file. Also, for each object project stores references to the objects that were inputs to the latest algorithm used to compute this object.

XML specifics

```
<?xml version="1.0" ?>
<bionic-project>
    <old id="rawimage_kwwt8snjt1z">
        <base_type>9_IplImage</base_type>
        <object_id>rawimage_kwwt8snjt1z</object_id>
        <source_id>/Users/tarasgalkovskyi/dev/bionic/data/raw_images_paul_test01/B73
            -1_10-18-10__01.jpeg</source_id>
        <type_name>rawimage</type_name>
        <value>./test-paul1/B73-1_10-18-10__01_rawimage_kwwt8snjt1z.jpg</value>
    </old>
    <data id="croppedimage_kwwt8yq0w6o">
        <base_type>9_IplImage</base_type>
        <history>
            <algorithm_name>cropping_anjali</algorithm_name>
            <algorithm_version>1</algorithm_version>
            <execution_time>367</execution_time>
            <inputs>
                <rawimage>croppedimage_kwwt8wlgq1y</rawimage>
            </inputs>
            <parameters>
                <crop_bottom>0</crop_bottom>
                <crop_left>0</crop_left>
                <crop_right>0</crop_right>
```

```
                <crop_top>1</crop_top>
                <rotate_heuristics>0</rotate_heuristics>
                <tolerance_vertical_1st_pass>1</tolerance_vertical_1st_pass>
                <tolerance_vertical_2nd_pass>1</tolerance_vertical_2nd_pass>
            </parameters>
            <type_name>croppedimage</type_name>
        </history>
        <object_id>croppedimage_kwwt8yq0w6o</object_id>
        <source_id>/Users/tarasgalkovskyi/dev/bionic/data/raw_images_paul_test01/B73
            -1_10-18-10__01.jpeg</source_id>
        <type_name>rawimage</type_name>
        <value>./test-paul1/B73-1_10-18-10__01_croppedimage_kwwt8yq0w6o.jpg</value>
    </data>
    <data id="croppedimage_kwwt8zg5kkp">
        <base_type>9_IplImage</base_type>
        ...
    </data>
</bionic-project>
```

Every data object in the project has corresponding xml "data" tag. XSD schema of the xml storage file follows:

```
TODO
```

7.4 Configuration format

Configuration for each algorithm and core is represented by distinct sections that do not mix. Each section consists of set of parameters: key-value pairs. Sections are identified by their ids that correspond to the corresponding algorithm name. So, one configuration file can contain at most one section per each distinct algorithm. If more than one configuration for specific algorithm is needed user is encouraged to override default configuration and provide additional configurations in separate files.

Except sections with configuration for algorithm, there are several dedicated sections: default algorithms and core configuration.

```
<?xml version="1.0"?>
<configuration>
    <core>
        <property name="image_format" value="jpg" />
        <property name="serialize_image_format" value="jpg" />
    </core>

    <algorithm_manager_defaults>
        <property name="thresholdedimage" value="adaptive_image_thresholding" />
    </algorithm_manager_defaults>

    <algorithm id="adaptive_image_thresholding">
        <property name="max_value" value="255"/>
        <property name="adaptive_method" value="CV_ADAPTIVE_THRESH_MEAN_C"/>
        <property name="threshold_type" value="CV_THRESH_BINARY"/>
        <property name="block_size" value="25"/>
        <property name="subtract_constant" value="-2.0"/>
        <property name="max_component_size_to_ignore" value="200" />
    </algorithm>
```

```
    <algorithm id="double_adaptive_thresholding">
        <property name="block_size" value="150" />
        <property name="drop_value" value="5" />
    </algorithm>

    <algorithm id="color_to_gray_image_converter">
        <property name="reverse" value="1"/>
    </algorithm>
</configuration>
```

7.5 Core configuration

This configuration section has name `core`. Using core settings it is possible to change the storage format of images being processed and computed.

- `serialize_image_format` Format for storing images in the project. Possible values are `jpg`, `tiff`, `bmp`
- `thread_pool_size` Number of threads to use for processing images. Should be positive. Best value for highest performance would be equal to number of physical cores

7.6 Default algorithms

This configuration section has name `algorithm_manager_defaults`. User can notify workflow builder about explicit choice for computing specific data types. So, for example user wants to use algorithm `adaptive_image_thresholding` to compute data of type `thresholdedimage`, to do that we need to provide such key-value pair in the proper configuration section.

```
<algorithm_manager_defaults>
  <property name="thresholdedimage" value="adaptive_image_thresholding" />
</algorithm_manager_defaults>
```

In such case, there will be no ambiguity when selecting path of workflow to compute `thresholdedimage` data type.

Chapter 8

Advanced: Writing Algorithms

8.1 Compiling sample algorithm

We recommend compiling and testing a custom algorithm to get familiar with the extensibility concept of GiA Roots. The *histogram thresholding* algorithm is provided in the *sample-algo* subfolder as part of the developer package.

To compile it you need to have a CMake tool (`http://cmake.org`). It can be downloaded precompiled for most of the OSes. You need to point cmake to the *sample-algo* folder, config and execute. If you are using command-line tool, you can execute right in the *example-histogram* folder *"cmake ."*. After that CMake should have generated project files for the IDE of your choice (or just makefiles).

If you have chosen makefiles, you should execute *make all* and two binaries will be generated – algorithm dynamic library (which should be placed in the *algorithms* subfolder if you want to test it) and the executable file. That executable file is standalone executable of only this custom algorithm. It assumes that input data is already stored in some dataset and can be loaded without executing any other algorithm. It will also provide you with any errors or suggestions that might affect your implementation of algorithm (if you conform to all the requirements posed by the framework). Compiled standalone executable will only work, if the working directory is the root folder of the development package (where the *giaroots-cmd* is situated).

IMPORTANT! If you are using Microsoft Visual Studio on Windows you should compile the algorithm and executable in the *Release* or *RelWithDbgInfo* mode. Otherwise it will not work within the framework. Also, you can debug the executable right in the IDE if you will choose *RelWithDbgInfo* mode and specify correct working directory in the project properties (same as where *giaroots-cmd* is situated).

IMPORTANT! If you are using XCode on MacOS you might be able to compile in any preferred mode (*Debug* or *Release*. Also, you can debug the executable right in the IDE if you will choose *Debug* mode and specify correct working directory in the project properties (same as where *giaroots-cmd* is situated).

8.2 Creating new algorithm

In the root folder of GiA Roots create a new folder with the name of your algorithm, suppose its name is "new_features". In this way, the root GiA Roots folder will contain the following folders: algorithms, include, jobs, lib, example-histogram, and new_features. Note that in future versions of GiA Roots all user's algorithms will be contained in one folder.

As a template, copy the files CMakeLists.txt and histogramthresholding.cpp from the folder "sample-algo" to the folder "new_features". Open CMakeLists.txt file and modify the following lines:

```
project(histogram-example) ---->   project(new_features)

set(SRC "histogramthresholding.cpp") ----> set(SRC "newfeatures.cpp")
set(ALGO_NAME "histogramthresholding") ----> set(ALGO_NAME "newfeatures")
...
```

Rename histogramthresholding.cpp file into newfeatures.cpp.

Open CMake. In the text field "Where is the source code" browse to your folder "new_features". In the text field "Where to build the binaries" browse to the folder "new_features/build" (build folder will be created automatically). Then press "configure", and "generate" in stated order. A new project file together with other files and folders will be generated in the folder "new_features/build/".

Open the project in your favorite programming environment. Modify newfeatures.cpp as you need. If there is now errors of coding it should compile, and a dynamic library for Release mode will be placed in the folder "new_features/build/Release". You can also use Debug mode to generate dynamic library and executable files which will be place in "new_features/build/Debug folder".

Copy dynamic library from the "new_features/build/Release" folder into algorithms folder (it makes sense to create subdirectory to easily distinguish your own algorithms from standard ones).

Go to jobs folder and create a new job file. You can use job files present in the folder jobs as a template.

Run *giaroots-cmd* executable to run job files.

Chapter 9

Legal notices

9.1 Copyright

All users must abide by the beta license testing agreement.

GiA Roots includes software developed by the OpenCV User Group.

9.2 Licenses

```
OpenCV - "a library of programming functions for real time computer vision."
http://opencv.willowgarage.com/wiki/
BSD License

Tinyxml - "a simple, small, minimal, C++ XML parser"
http://sourceforge.net/projects/tinyxml/
Licensed under zlib

QT - "a cross-platform application and UI framework"
http://qt.nokia.com/products/
Copyright (c) 2008-2010 Nokia Corporation and/or its subsidiaries.
Licensed under LGPL v2.1.
```